Volume I

Ada Implementation Guide

Software Engineering With Ada

April 1994

DEPARTMENT OF THE NAVY

Naval Information Systems Management Center

Acknowledgements

The following DON personnel contributed their valuable time, insights, and perspectives in developing this guide. Their contributions and hard work were invaluable and are greatly appreciated.

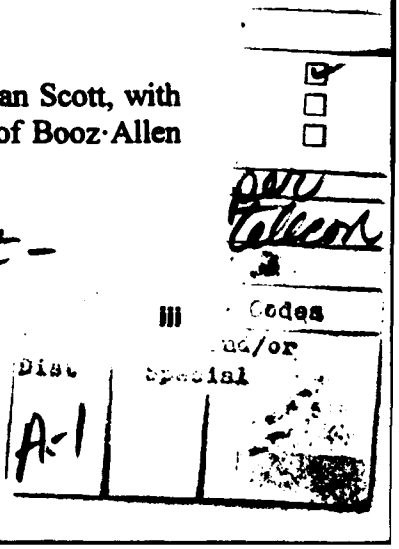
Chair:	Ms. Antoinette Stuart, NISMC
Deputy Chair:	CDR Martin Romeo, SPAWAR

Major George Bedar	USNA
Mr. Currie Colket	AJPO
Mr. Tom Coyle	NAVAIR
Major Gerald DePasquale	MARCORCOMTELACT
Mr. Les Dupaix	USAF/STC
Mr. Greg Engledove	NAVSEA
Ms. Donna Fisher	NRaD
Mr. Charles Flemmings	NSWC Dahlgren
Ms. Patricia Grandy	NARDAC, San Francisco
Mr. Ron House	NUWC Newport
Mr. Chuck Koch	NAWC-AD-WAR
Dr. Yuh-jeng Lee	NPS
Ms. Joan McGarity	NCTC
Mr. John McLaurin	NADEPJAX
CDR Lindy Moran	NCTC
Major John Myers	USNA
LT Don Needham	USNA
Ms. Tricia Oberndorf	NAWC-AD-WAR
Mr. Mike Rice	NAVSEA
Mr. Ramon Rivera	FMSO
Mr. George Robertson	NRaD
LCDR Jean Shkapsky	BUPERS
Mr. Barry Siegel	NRaD
Major J. Spegele	MCSA/DITSO-KC
Mr. Charles Stokes	NAVAIR
Mr. Hank Stuebing	NAWC-AD-WAR
LCDR Anne Sullivan	DISA/JIEO/TEWS J
Mr. Tim Walton	FMSO

We also wish to express our thanks to the document coordinator, Ms. Susan Scott, with support from Ms. Salvi Mugol, and the editor, Ms. Madeline Nevins, all of Booz·Allen & Hamilton Inc.

St #A, auth: AJPO (Mr Currie Colket - 602-3968) telecon, 7 July 94

CB



Contents

VOLUME I

VOLUME II

List of Figures and Tables

Figures

Tables

Section 1
Introduction

Public Law 102-396, Section 9070 of the Department of Defense (DOD) Appropriations Act, 1993, enacted on 6 October 1992, requires that, "where cost-effective, all Department of Defense software shall be written in the programming language Ada . . . ." The Department of the Navy (DON) prepared this second edition of the *Ada Implementation Guide* to help Program Managers and their staffs to implement this law. (For the purposes of this document, the terms Program Managers and Project Managers are synonymous.)

New additions to this guide include the following:

- Software engineering and Ada training requirements
- Information on standard Ada bindings to commercial application software
- A section on the way Ada facilitates the application of software engineering principles
- Integration of Ada and new emerging technologies (e.g., open systems architecture, software reuse, computer-aided software engineering, reengineering)
- A DOD/DON Software Policy Matrix that includes policy descriptions and related policies.

1.1 SCOPE OF Ada USE

Ada was developed to control the proliferation of programming languages, establish a standard programming language for DOD, and reduce DOD's cost to maintain software for its mission-critical systems. Congress has recognized Ada as the standard for all DOD software application development and has mandated its use unless an alternative approach can be demonstrated to be cost-effective over the application life cycle. As a matter of DOD policy, all new systems and major software upgrades are subject to this mandate. The mandate to use Ada applies only to software that DOD is developing and must support and maintain throughout the life cycle. Organizations are encouraged to use Commercial-Off-The-Shelf (COTS) software to fulfill operational requirements as development tools and even support libraries within an application. The language used for these artifacts is immaterial because DOD does not maintain the software. When DOD does maintain the software, however, the language does matter, and DOD must have an infrastructure

to provide cost-effective maintenance and supportability of the software over the system life cycle to include Post-Deployment Software Support (PDSS). Use of a single DOD language to support this infrastructure is not a new concept: DOD organizations have required the use of particular languages since the 1960s; in the mid-1970s, the use of a smaller set of languages was mandated for DOD applications; in 1985, the use of Ada was mandated for a class of DOD systems; and in 1990, the Congressional mandate made Ada the standard language for all DOD systems.

This guide addresses Ada policy implementation for two broad classes of DON systems: Mission-Critical Computer Resource (MCCR) systems and Automated Information Systems (AIS). The term MCCR is used in this guide to denote the class of systems managed under the DODD 5000 series of instructions. The term AIS is used to denote those systems managed under the DODD 8000 series of instructions. The communities of program managers who manage these systems have distinctly differing environments for implementing Ada policy.

The MCCR community consists of organizations that work on computer resources critical to the conduct of the military mission of the DON, including those for all tactical and strategic weapons, communications, command and control, cryptologic activities related to national security, and intelligence systems that directly support military operations. Such systems often are embedded. The MCCR community has been developing software systems in Ada since 1985; however, most systems development has been contracted to the private sector.

The AIS community comprises organizations that work on business computer resources that are not mission critical, including all administrative, logistics, financial, personnel, and work load planning systems. Such systems may operate on microcomputers or mainframe computers in a stand-alone or networked mode. The AIS community has developed its software systems in a variety of High Order Languages (HOLs) by using inhouse DON personnel. Although relatively few AIS applications systems have been developed in Ada, the number is increasing as activities and claimants are developing Ada expertise.

Whether the system is MCCR or AIS, Ada must be used during the concept exploration and definition, demonstration and validation, engineering and manufacturing, development, and production/deployment phases of a system and for both application and support software. In addition, Ada must be used for major modifications to existing systems and for systems that involve integrating components into systems that incorporate commercial and other nondevelopmental software. To use an alternative programming language, a waiver must first be obtained from the appropriate waiver authority; the waiver must include an economic analysis that

clearly delineates that an alternative approach will be significantly more cost-effective.

1.2 EXPERIENCE OF Ada USE

Operation Desert Storm illustrates why DOD's use of a single HOL is appropriate. During this conflict, the Air Force's Theater Display Terminal (TDT) had been deployed to Israel to warn of Iraqi SCUD attacks. The TDT is a deployable missile warning system written in Ada for a Sun 3 UNIX environment. On 11 January 1991, Israel identified a new system requirement for the TDT operational program, namely, to identify the country of origin for a missile launch. Knowing the country of origin was considered important in formulating the tactical response. To Israel, it made a difference whether a SCUD was launched from Iraq or from some other country. By 13 January 1991, an existing country-of-origin algorithm in Ada and an Ada geopolitical database were found. On 13 January 1991, these artifacts were integrated into a developmental system; on 14 January the enhancement was integrated into an operational system. The software was flown to theater and installed for use on 15 January. On 16 January, only 5 days after the requirement was identified, the new capability was ready to detect Iraqi SCUD attacks on Israel. The capability to support this new mission requirement rapidly was possible only because the TDT, the country-of-origin algorithm, and the geopolitical database were in Ada. Integration of these artifacts was possible only because of the Ada package, which provided clean logical interfaces between each artifact. Had Ada not been used, the cost to support the new requirement would have been substantial, and the enhancement would not have been fielded by the end of Desert Storm.

Today, use of Ada enables DOD, government, and commercial organizations committed to Ada to achieve a higher-quality product at reduced life-cycle costs. Just as Ada use makes sense to DOD for sound economic reasons, its use also makes sense to the commercial world for the same reasons. Among the companies that are using Ada are Digital Equipment Corporation, Boeing, Motorola, and Rockwell. Boeing's 777, the Federal Aviation Administration's Advanced Automation System, and the National Aeronautics and Space Administration's Space Station are non-DOD projects committed to the use of Ada. These organizations are openly embracing Ada, not because of the Ada mandate, but because using Ada to produce high-quality software for large, safety-critical applications is a sound economic business decision.

Boeing is an excellent example of a commercial organization committed to using Ada. The Boeing 777, with its estimated 10 million lines of code, will be one of the largest software projects ever undertaken. Boeing expects to take advantage of software reuse by reusing 2 to 4 million lines of existing code. Boeing also used Ada on the recent modification to the Boeing 747, which is now flown extensively by the

airlines. Boeing is committed to the use of Ada for this system because of the quality, cost, and schedule benefits provided when Ada is used in combination with modern software engineering practices.

Data obtained by comparing the B-52 Offensive Avionics System (OAS) Modification in 1979–1980 and the F-22 Advanced Tactical Fighter (ATF) Demonstration Evaluation (DEMVAL) in 1990 helped Boeing decide to commit to Ada on the 777. The B-52 OAS was written in 120,000 lines of JOVIAL source code. Originally, about 15 flights had been scheduled to test the software. During flight testing, more than 800 problem reports were identified, and approximately 80 flights were required to complete the software testing. Testing was unnecessarily complicated, expensive, and problematic because, on average, 30 to 50 patches were entered into the tested software. In contrast, the F-22 ATF was written in 250,000 lines of Ada source code. This represented far more than twice the complexity of the B-52 OAS because Ada is capable of dealing with high-level abstractions. Thirty-one flights had been scheduled to test all the complexities of the flight software. Only eight problems were identified against the flight software. This represents an improvement of two orders of magnitude from the B-52 OAS to the F-22 ATF DEMVAL. Testing was facilitated because a clean compile was available as required because of Ada's support of separate compilation. These statistics are even more impressive given that seven different contractors developed the ATF software and a team of programmers integrated it at the flight test site the week before the first flight. Certainly the advances in software engineering and tools between 1980 and 1990 played a role in these statistics. The support of these software engineering advances and tools for the Ada language was an important factor in Boeing's commitment to Ada for the Boeing 777.

1.3 DOCUMENT SCOPE

This document is intended to guide Program Managers in using Ada in systems development throughout the various phases of acquisition and to promote the use of sound software engineering principles, concepts, and processes in systems engineering. All levels of management, technical personnel, and the systems development community should refer to the information contained herein.

The intent is to update this document biennially. In keeping with the spirit and intent of the document, all users of this guide are encouraged to share their experiences and knowledge with the Ada community. Therefore, if a reader finds that significant areas are not covered in this document, we request that he or she provide feedback to the document update process so that the next user will have better information. (See Appendix A, Section A.1.1, DON Ada Representative address.)

1.4 CONTENT AND ORGANIZATION OF THIS DOCUMENT

This two-volume document discusses the use of Ada for software system development in the DON, and its contents apply to both AIS and MCCR communities. Volume I contains eight sections, a glossary, a list of acronyms and abbreviations, a reference list, a bibliography, and an index. Section 1 provides information on the content of this document. Section 2 describes DON policy with regard to Ada use. Section 3 contains specific guidance on incorporating Ada into all phases of life-cycle management—from program planning through post-deployment maintenance. In Section 4, the environments that support development and maintenance of Ada application software are described. Section 5 addresses the concept of software engineering and identifies several Ada features that are important in supporting software engineering. Section 6 presents a series of lessons learned relevant to the software development process, and Section 7 highlights what DON Program Managers can expect in the future. Section 8 contains information on recommended training in Ada and software engineering for DON software professionals.

Volume II contains several appendixes that provide valuable information to supplement the guidance contained in Volume I. Appendix A describes additional resources that will be helpful to Ada Program Managers and users. As noted, Appendix B provides a matrix of DOD/DON software policy. The other appendixes expand on issues discussed in Volume I, and they are referenced where appropriate.

Section 2

Ada Policy

This section summarizes the Department of the Navy (DON) policy on Ada use in Automated Information Systems (AIS) and Mission-Critical Computer Resources (MCCR) programs.

2.1 POLICY DIRECTIVES

Public Law 102-396 requires that, "notwithstanding any other provision of law, after June 1, 1991, where cost effective, all Department of Defense software shall be written in the programming language Ada, in the absence of special exemption by an official designated by the Secretary of Defense." Figure 2-1 provides the directives and instructions that serve as the framework for DON implementation of this law.

2.2 POLICY RATIONALE

The thrust of the Department of Defense (DOD) computer programming language policy has been to limit the number of different computer languages, including dialects and support tools, associated with the application software maintained by DOD. DOD estimates that maintenance accounts for 60% to 80% of software life-cycle costs; therefore, DOD emphasis is on one High Order Language (HOL) and a limited set of standard HOLs.

The selection of Ada as the "single, common, approved standard HOL" for DOD systems was based on the inherent software engineering features of the Ada programming language. These software engineering features lead to software programs that are better structured, less error prone, and more easily maintained. In addition, these features facilitate reuse and system reengineering. Ada and Ada Program Support Environments (PSEs) enable DOD to deal effectively with the programmatic and technical challenges posed by the increasing number and complexity of software-intensive systems.

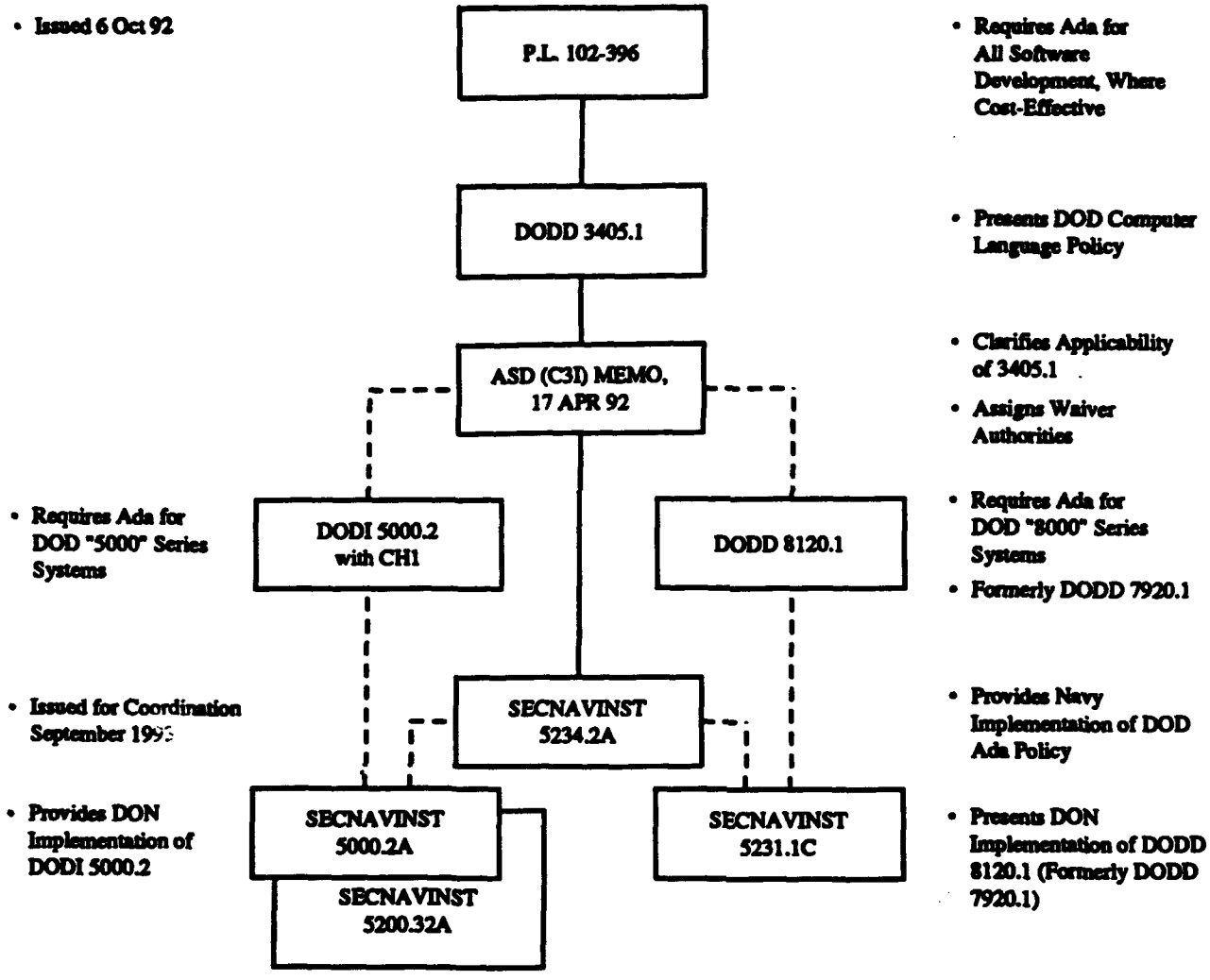


Figure 2-1. DON Directives and Instructions for Implementing Public Law 102-396

2.3 POLICY DESCRIPTION

Table 2-1 provides a description of the essential features of DON Ada policy as planned to be issued in SECNAVINST 5234.2A. The table has been prepared with the expectation that the SECNAVINST 5234.2A will be issued at about the time this guide is promulgated. Although the table has been constructed to be consistent with SECNAVINST 5234.2A, in the event of conflicts, the provisions of SECNAVINST 5234.2A take precedence. Appendix B provides additional information on DOD/DON instructions related to Ada, software engineering, and life-cycle management policy.

Table 2-1. DON Ada Policy Implementation Matrix

AREA	Ada POLICY DESCRIPTION	COMMENTS
Scope/ Applicability	This instruction applies to all systems and software that: (a) are managed under SECNAVINST 5200.32A, or (DOD 5000 Series) (b) are managed under SECNAVINST 5231.1C, or (DOD 8000 Series) (c) are part of Science and Technology (S&T) programs under direction and oversight of the Office of the Chief of Naval Research (OCNR). This instruction does not apply retroactively to the following: • Systems that have entered production and deployment (passed Milestone III) for (a) above. • Systems managed under (b) above that have passed Milestone II as of 1 June 1991 • Systems managed under (a) above for which a documented language commitment was made in accordance with previous policy. This policy does apply at the first major software upgrades for these systems. This policy supersedes SECNAVINST 5234.2.	Major software upgrade is defined as "a change to the system architecture which would result in a cummulative one-third modification to a computer software configuration item (DOD-STD-2167A) or a subsystem specification (DOD-STD-7835) within any five year period as measured in compileable source lines of code. Modification includes addition, deletion, or change exclusive of routine maintenance."

Table 2-1. DON Ada Policy Implementation Matrix (continued)

AREA	Ada POLICY DESCRIPTION	COMMENTS
Existing Operationally Fielded Software	Ada is not required for: Software maintenance limited to error correction and modifications for portability of existing software.	Reuse or upgrade of operationally fielded software for new acquisition programs or major software upgrades is addressed below.
Nondeliverable Software	Ada is not required for the development of nondeliverable software, as defined in DOD-STD-2167A, does not require Ada.	Rationale: By definition, nondeliverable code will not be delivered to or maintained by DOD.
COTS Software	· No waivers are required for the use of COTS software, including operating systems, utilities, libraries, self-contained applications programs, and vendor update implementations. The COTS software is not to be modified in function or maintained by the goverment.	Use of COTS software is encouraged and recommended.
Reuse of Ada code	No waiver is required.	Reuse of Ada software is encouraged and recommended.
Reuse and Upgrade of Existing DOD- and Government-maintained Software	No waivers are required for reuse of modification of existing operationally fielded non-Ada software, as long as the change to the reused or modified software is less than a major software upgrade. Changes to existing software that constitute a major software upgrade must be done in Ada or else a waiver is required.	Major software upgrade is defined above.

Table 2-1. DON Ada Policy Implementation Matrix (continued)

AREA	Ada POLICY DESCRIPTION	COMMENTS
Advanced Software Technology (AST)	No waivers are required for the use of a commercially available off-the-shelf AST that is not modified or maintained by the Government. *Note: Use of SQL (ANSI, FIPS 127-1) with compliant COTS DBMS's for binding to Ada host applications is an Ada policy compliant approach. The software engineering approach for constructing such DBMS applications is described in: - SEI 88-MR-9 - SEI 89-SR-14 - SEI 90-TR-26	ASTs include life-cycle support tools, programming support environments, non-procedural languages (4GLs), and modern DBMSs. Although the native commands associated with ASTs may be used for ad hoc query transaction, any programs written for use with the AST or HOLs, or used in conjunction with AST must use Ada or Ada bindings.
Waiver Authority	As specified in SECNAVINST 5234.2A.	

Section 3

Implementation Guidance

To capitalize on the benefits of software engineering with Ada, the Program Manager must carefully plan for its use throughout the program life cycle. Department of the Navy (DON) programs have multiple stages: planning, acquisition and/or development, test and evaluation, and Post-Deployment Software Support (PDSS). It is critical that the Ada software engineering process be assimilated into the project's systems engineering activities at the earliest possible time. This combination of systems engineering and Ada will provide the Program Manager with tangible benefits such as higher quality software, fewer system integration problems, ease of post-deployment maintenance, and software reusability—all of which result in reduced life-cycle costs.

This section provides guidance to help Program Managers effectively engineer and incorporate Ada into their programs. It discusses the interrelated elements of program planning, acquisition planning, systems engineering, and risk management. It is important that the Program Manager's acquisition and program planning reflect systems engineering (i.e., an integrated approach to hardware and software) from the first stages of the acquisition process.

This section assumes that the Program Manager is well versed in the areas of software and systems engineering and the appropriate Department of Defense (DOD) and DON policy and directives.

3.1 PROGRAM PLANNING

To effectively incorporate Ada into systems development, the following areas must be addressed:

- Organizational structure
- Cost estimation
- Resource planning.

The Program Manager is responsible for ensuring that program planning thoroughly addresses these key areas. It is critical that the acquisition strategy fully integrate software and hardware systems development as early as possible in the process.

3.1.1 Organizational Structure

Ada software development and maintenance efforts are compatible with all organizational structures (Archer, 1991). The use of Ada does not restrict project size, functional organization, combinations of organizational structures, or the

function of the organization concerned. As a matter of good business and systems engineering practices, Program Managers should:

- Establish their Program Management Organization and organizational interfaces including:
  - Operational users
  - Test and evaluation organizations
  - Software development organizations (contractor and organic)
  - Organic support organizations (e.g., warfare centers, Life-Cycle Support Activities [LCSAs], design programming activities, Software Support Activities [SSAs], and In-Service Engineering Agents [ISEAs]).

- Ensure that Ada, software engineering, domain-specific architecture, and engineering expertise are available within their management and support organizations.

- Ensure that organic support activities have established internal management practices for improving their software engineering capabilities. These management practices should include provisions for introductory and continuing Ada and software engineering training, incorporation of Software Engineering Institute (SEI) software process improvement plans, and institution of Software Engineering Process Groups (SEPGs).

- Promote the use of Ada and ensure that Ada policy requirements are known and understood within the management and support organization.

3.1.2 Cost Estimation

Department of Defense Directive (DODD) 5000.1, Department of Defense Instruction (DODI) 5000.2, DODD 8120.1, and DODI 8120.2 require that system life-cycle cost estimations include cost forecasts for development, procurement, and support. Program Managers are responsible for performing and submitting developmental and support cost estimates for their systems. These cost estimation data are used to prepare Program Objective Memorandum (POM) funding estimates, Logistics Requirements Funding Plans (LRFPs), Cost and Operational Effectiveness Analyses (COEAs), and other system life-cycle milestone decision documents. They serve as the Government estimate for contract evaluations.

As DON systems have become increasingly software intensive, software costs have become a significant factor in overall system life-cycle costs. Software maintenance is the single largest life-cycle cost for Government computer systems, with software development costs coming next. Government computer systems have typically been delivered over cost, behind schedule, and lacking in quality and maintainability.

It is crucial that accurate software cost estimations be performed to provide a basis for the Program Manager to plan the organization, schedule, and resource requirements needed to sustain the software from the initial software development throughout the system's life cycle.

One of the most challenging tasks in software management is to accurately estimate the resource requirements and time needed for a software development project. Section 6, Lessons Learned, notes that, on some projects, initial planning estimates for software development understated the resource requirements and time needed and severely affected actual system costs.

The three generally accepted approaches for estimating the resources and schedule needed for software development are as follows:

- Comparing the proposed project with a completed project of the same type the cost and schedule of which are known
- Dividing the overall development effort into several smaller tasks (often using a multilevel Work Breakdown Structure [WBS]) and adding up the estimates for each of these tasks. (See Software Work Breakdown Structure, MIL-STD-881B, and the draft Handbook, MIL-HDBK-171.)
- Using a software cost and schedule model that calculates the resources and time needed as a function of other software parameters (e.g., size, complexity, function points).

Regardless of the estimation approach used, an organization with more experience in developing software of the same size and complexity for the same type application will produce more accurate estimates for a new project. Unfortunately, estimates based on experience with small and less complex software developments are almost always unreliable because this experience cannot be applied to larger, more complex software in a similar application area. This difference results partially from an exponential relationship between software size and development effort. Experience in one application area does not always transfer well to other application areas.

Many software cost estimating methods have evolved over the years. These methods take the form of either analog or parametric models and may be implemented manually or by using automated software cost estimating tools. Most of the current models require size estimates of either the lines of source code or function points as input. The resulting estimates for cost, effort, and schedule are highly dependent upon the accuracy of the initial size estimates. A few software tools, such as those

based on function point analysis (Jones, 1991), do not require project size as an input.

Experience has shown that the best software cost estimates have been based on an application of two or more methods/models by experienced estimators. The Naval Center for Cost Analysis (NCA) can provide:

- Expert advice on cost estimation
- Assessments of software cost estimation tools being considered for use by DON programs (independent)
- Lessons learned from cost estimating
- Advice on estimating software development and maintenance costs.

Appendix A, Section A.1.1, provides points of contact and addresses of organizations involved in cost estimating, and Appendix D provides additional information on cost estimating.

3.1.3 Resource Planning

The major areas for resource planning are as follows:

- Personnel and training
- Schedule and time
- Development, test, and operational environments (hardware and software)
- Life-cycle documentation
- Software development process improvement
- Metrics
- Data management and analysis
- Life-cycle maintainability (PDSS)
- Information sources.

The subsections below discuss these areas.

3.1.3.1 Personnel and Training

The Program Manager must ensure that software engineering and Ada training is conducted at all levels of the organization (management and technical). Project success depends on the ability of people who have the knowledge and skills needed to perform the system and software engineering functions, develop the process control mechanisms, collect critical data and metrics, and analyze data to determine status and trends.

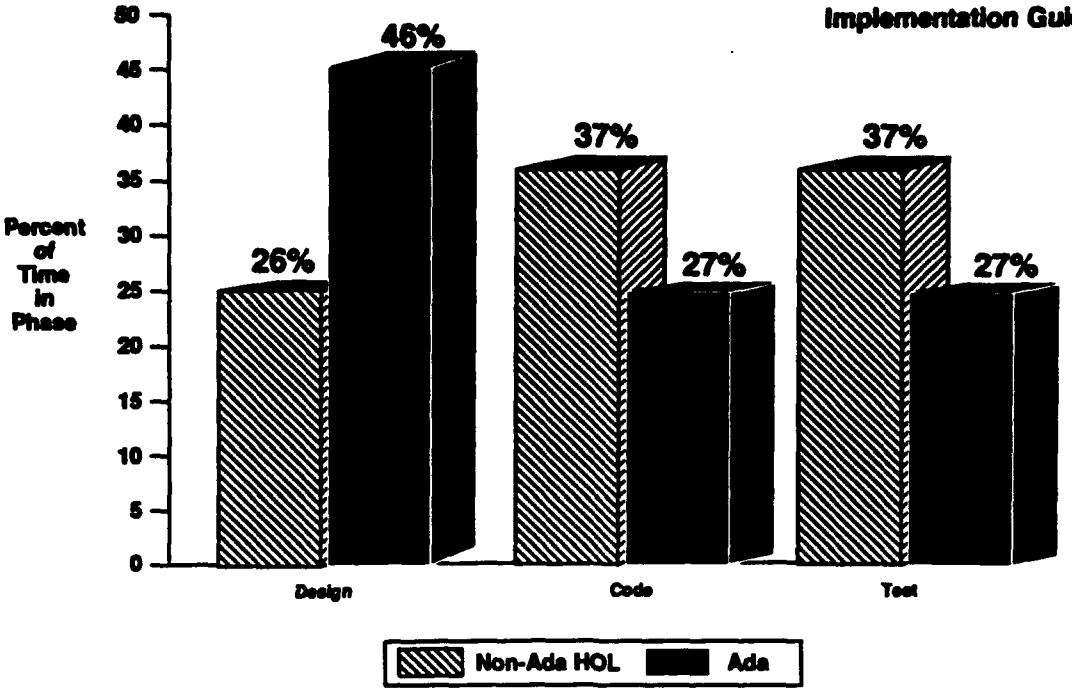
Most computer science graduates who are trained in Ada and software engineering principles can quickly adapt this knowledge to a specific application. Staff members who have spent years working with traditional programming languages and production processes may require additional training and mentoring. With this additional training, these individuals can become a valuable part of the program by using their inherent knowledge of corporate functions and experience in the application domain. Training should be given "just in time"; that is, personnel should be assigned to a project where they can immediately capitalize on their training. Consideration should also be given to conducting continuing education at local universities, via membership in software professional societies, and at software engineering and Ada symposia. Appendix A, Sections A.1.6 and A.3.4 provide information on Ada professional societies. Section 8 in this volume provides detailed descriptions of the contents and recommended length of the courses available.

3.1.3.2 Schedule and Time

Time is the Program Manager's most important consumable resource. Program Managers must identify milestones and select and use a realistic process for measuring progress towards achieving those milestones. By using a variety of measures to track progress and plan contingency actions, Program Managers can best mitigate schedule risk. Although cost tracking mechanisms, such as WBSs, can be useful, their exclusive use does not constitute a valid measure of progress.

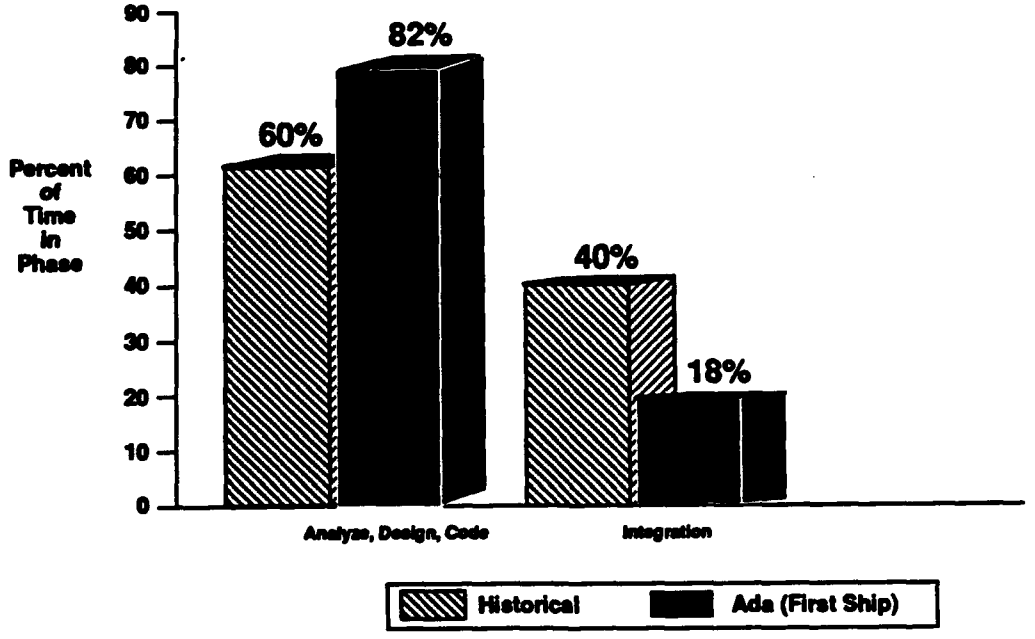
Program Managers should be aware that the allocation of resources to a schedule is different for the Design, Coding, and Testing Phases of a project when using Ada. In general, Ada projects require more time for design and less for coding and testing than non-Ada projects because errors detected in the Design Phase are less expensive to correct. Figure 3-1 depicts the results of a National Aeronautics and Space Administration (NASA) study that analyzed the time spent on the various phases of Ada and non-Ada projects conducted from 1983 to 1993. As shown in Figure 3-1, non-Ada projects typically required less time for the Design Phase and more time for Coding and Testing Phases.

Time invested in the Design Phase reduces the time required for the Coding and Testing Phases, and, according to data provided by NobelTech (now Celcius Tech) of Sweden, it also reduces time needed for integration. Figure 3-2 shows the reduction in time achieved by NobelTech for integration. In the past, integration represented about 40% of a project's total time. When NobelTech used Ada on its



Source: NASA GFSC, 1983-1990

Figure 3-1. Development Time for Software Engineered Projects



Source: NobelTech, Sweden

Figure 3-2. Reduction in Integration Time

first development project, the integration time was only about 18% of the total time. Time is especially important during system integration when many subsystems must come together, often for the first time. Delay in producing a critical subsystem could seriously affect the progress of other subsystems. Use of Ada helps to reduce integration problems because its language features help provide a clear interface between subsystems. Appendix L, Section L.1, contains additional information on the Ada package feature that provides this clear interface.

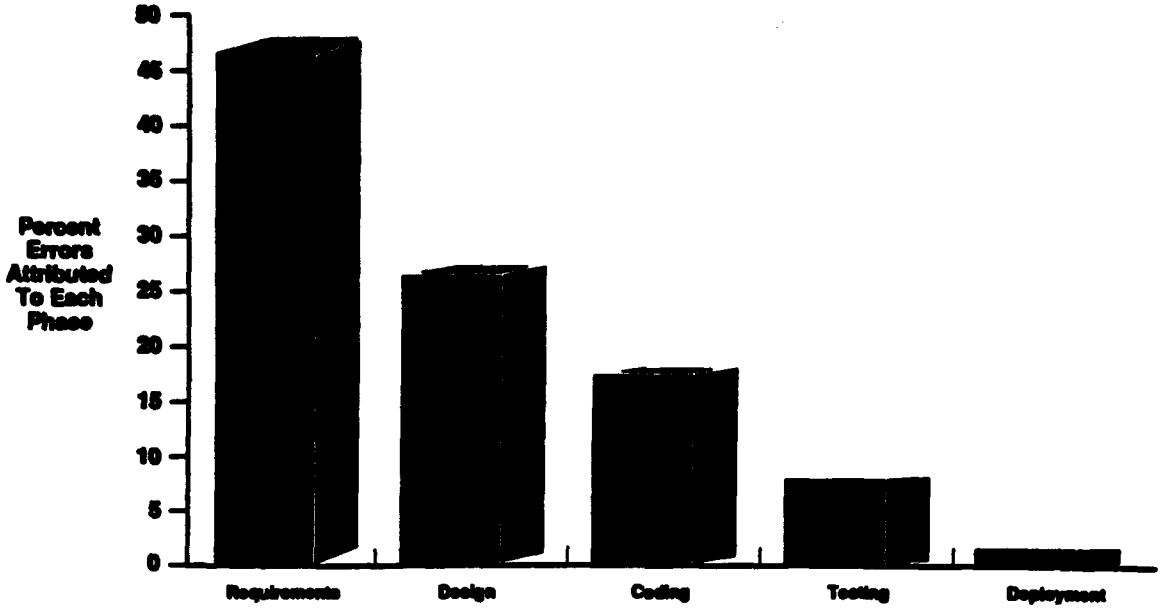
During schedule planning, it is important to realize that time invested in both the requirements and design phases will provide significant benefits to the overall system life-cycle costs. Errors in the Requirements and Design Phases are typically high and are the most costly to correct. Figure 3-3 shows the source of software errors as reported in the Communications of the Association for Computing Machinery (ACM), (Association for Computing Machinery, 1984). As the figure shows, more than 75% of errors result from the Requirements and Design Phases. Figure 3-4 shows the relative cost to correct these errors based on a study from AT&T, Bell Laboratories, 1988. A requirement error detected in deployment is almost an order of magnitude more expensive to correct in the Deployment Phase than in the Requirements Phase. An error in design is about 25% less expensive to correct in the Design Phase than in the Deployment Phase.

As illustrated in the NASA, NobelTech, and other studies, the use of Ada and software engineering results in the early detection of design and requirement errors, thus reducing overall life-cycle costs. Figures 3-3 and 3-4 also imply a Program Manager should invest additional time in developing, analyzing, and validating requirements. Many projects would have benefited from a more rigorous Requirements Phase.

3.1.3.3 Development, Test, and Operational Environments

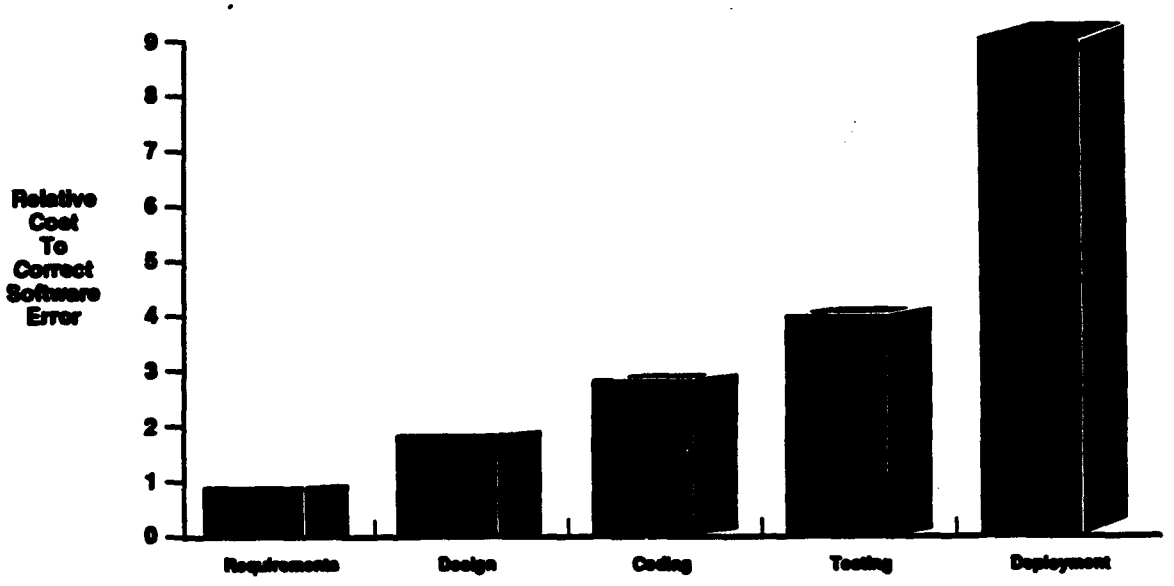
Software and software development methods within DON must keep pace with the latest development and test technology. The Program Manager is responsible for ensuring that new technology is identified, evaluated, and integrated into the acquisition process when it provides a better system solution. To minimize program risk, only those technologies that have been fully proven and demonstrated should be adopted for Full-Scale Development (FSD).

Use of the best-of-industry tools and a highly trained, competent staff ensures the highest productivity and quality. Profiles of DOD projects show that, historically, for every line of operational code, there have been at least four lines of support software (Boehm, 1981). Traditionally, this approach has increased the hidden costs of a project because support functions were either performed in a labor-intensive manner



Source: Communication of ACM, January 1984

Figure 3-3. Source of Software Errors



Source: AT&T

Figure 3-4. Relative Cost to Correct Software Errors

or additional unplanned costs were incurred in tool development. The diversity of commercial project support tools available to assist in the generation, documentation, configuration control, and maintenance of operational code should minimize the need to develop project-specific support tools.

The judicious selection of such tools can enhance software development, maintenance, and productivity; however, the Program Manager should ensure:

- The project support tools selected are formally demonstrated and tested before committing to their extensive use
- Software methodologies used in the support tools are consistent with practices used in the supporting software development or maintenance activity
- Support tools are used consistently from the highest level of management through the subdeveloper level
- Interface and data interchange incompatibilities among various tools are resolved.

The Program Manager also should contact the U.S. Air Force Software Technology Support Center (STSC) for information and evaluation data concerning applicable Software Engineering Environment (SEE) tools. Appendix A, Section A.1.1, provides information on the STSC.

Modern systems are becoming more complex and frequently test the limits of the system development process. The software development environment should be robust and flexible to accommodate unforeseen requirements. Therefore, managers of development activities should evaluate development tools and select those that are best for their application. The tendency to procure host or target configurations that minimally support the requirements of the software development process must be overcome. It is important to provide the software development activity with sufficient processing and memory capacity to allow it to accomplish its task. Memory and processing capacity are relatively inexpensive when the value of functions performed and productivity gains achieved by software tools are considered. Program Managers must also realize that an adequate number of trained professionals will be required to use and maintain these tools and environments.

3.1.3.4 Life-Cycle Documentation

Documenting and disseminating the standards and procedures needed to manage the development process are necessary to ensure discipline and consistency in the development and maintenance phases of the software system life cycle. Project

planning documentation must identify the applicable standards and procedures to be used during the project's life cycle. Section 2 identifies the requirements and instructions that describe life-cycle management and documentation standards.

These standards are meant to be tailored for each individual program and will vary from program to program. MIL-HDBK-287 contains tailoring guidance for DOD-STD-2167A, but no equivalent tool is available for DOD-STD-7935A. The Program Manager should realize that time spent up front in tailoring documentation requirements to the program's specific needs can substantially reduce total life-cycle costs.

3.1.3.5 Software Development Process Improvement

The basic principle of system process management is that a development process should be under statistical control. Statistical control means valid process indicators have been identified and measured so that statistical monitoring of this indicator, over time, will accomplish the following:

- Identify any deviation of the process from normal operations
- Serve as a long-term basis for process improvement
- Produce desired results within anticipated limits of cost, schedule, and quality.

Although statistical control is well known in the manufacturing arena, its use as a tool in the software development process is still in its infancy. Few process indicators have been identified that reliably contribute to long-term process improvement.

As a step toward providing guidance to organizations on how to gain control of software development and maintenance processes and evolve toward a culture of software excellence, the SEI has developed the Capability Maturity Model (CMM), which is described in Appendix C. To help apply the CMM, the SEI has developed two distinct evaluation techniques: Software Process Assessment (SPA) and Software Capability Evaluation (SCE). These techniques can help the Program Manager improve in-house software development and maintenance capabilities and can provide a basis for selecting qualified software developers.

3.1.3.5.1 Software Process Assessment

SPA is a self-diagnostic and improvement-initiating activity that helps software organizations launch effective process improvement programs. SEI or its licensed vendors provide SPA training. Appendix A, Section A.1.1, contains information on how to contact the SEI.

Based on examination of representative projects, SPA produces a baseline of the organization's process maturity. These projects may be old or new, large or small,

and they may use either the latest languages and methods or established languages and methods. An effective SPA will provide software development organizations with an indication of the current state of their software processes, the high-priority software process issues facing the organizations, and the actions needed to start process improvement.

3.1.3.5.2 Software Capability Evaluation

SCE is an audit technique that examines an organization in light of the contract to be awarded. Government acquisition organizations use SCE to help identify the capability of software organizations during source selection and to monitor contractor capability during contract performance. The software development organization provides candidate projects for evaluation by the acquisition organization as an indicator of its ability to perform on the contract. Program Managers should assess the maturity level of contractors and in-house software developers before beginning software acquisition or development.

3.1.3.5.3 Process Control

Program Managers must ensure that reporting mechanisms and procedures are in place to regularly review the status of development and that a method exists for assessing adherence to plans and controlling risk. The resolution of critical problems must include inputs from the end user, Program Managers, and developers.

Program Managers should institute the use of an SEPG as the primary means of process improvement and control. The SEPG facilitates software process definition and improvement efforts. This should also be required of both Government and contractor software development organizations. Program Managers should have on their staffs an SEPG member who will help the Program Manager work with the development organization's SEPG staff to coordinate, monitor, and facilitate all contract software process improvement initiatives, including SCEs, SPAs, metrics collection, and reporting.

Standardization of the metrics process is especially critical to allow analysis that will reveal the cause rather than the side effects of the problem. Formal record keeping and accurate traceability of top-level requirements down through coding, testing, and execution are vital to the analysis of system problems. The localization and modularity features of Ada enhance its capability to support traceability through assignment of requirements to a single code entity.

3.1.3.6 Metrics

Software metrics refer to the measurement of pertinent software process and product parameters. For example, during certification of Reusable Software Components (RSCs), use of a "reusability" metric provides an indicator of the level of effort

required to reuse the RSC in a new development. The reusability metric becomes a discriminator that helps managers choose between two or more RSCs that meet needed functional requirements. Selecting the RSC with the highest reusability score saves time and money.

Integrating metrics into the development process enables an objective assessment of development progress, resource expenditure, and technical product quality. Metrics must be identified, qualified, and applied at the beginning of the acquisition and systems engineering processes and maintained throughout the life cycle. Costs associated with the collection and analysis of metrics must be carefully considered in selecting appropriate software metrics. The need for metrics is discussed in greater detail in SEI's *Software Measurement Concepts for Acquisition Program Managers* (Rozum, 1992).

The Program Manager must ensure that a software metrics implementation plan is in place before contract award and that an experienced support organization is identified to implement the plan. This organization can initiate assessment activities during source selection by working with Request for Proposals (RFP)-specified and contractor-proposed software metrics. The implementation plan should provide for the following:

- Metrics should address specific issues of interest to the Program Manager. Many issues are common across programs (e.g., cost and schedule, personnel resources, Source Lines of Code [SLOC], defects). A standard set of metrics applicable across major programs should be identified as a core set from which the project-specific metrics program is built. The program should be flexible and expandable to address additional issues that may be derived from the analysis of the core set or other program concerns. These additional issues will depend on program size; acquisition strategy; the developer's process; cost, schedule, and technical risks; and changes in these characteristics throughout the life cycle.

- Because metrics are linked to a software process or product, they need to be collected on a planned basis throughout the system life cycle.

- Metrics should be defined and used consistently to provide a critical foundation for meaningful comparisons (e.g., planned versus actual costs, differences between configuration items).

- Metrics should be collected automatically when possible to ensure accuracy, consistency, and timeliness. On-line access to developer data in electronic

format should be provided throughout the life cycle. Developer data should be treated as proprietary to prevent misuse.

- Metrics should be analyzed and interpreted to assess achievement of milestones and quality of software products. Raw metrics can be misleading; to be useful, they must be interpreted with respect to the overall software process.

- Results of metrics analysis activities must be related to and used by decision makers. The quantitative results of software metrics analysis can be used as an indication of the overall health of the program.

To establish an effective metrics program, the requirement for metrics must be formally established in contractual documents. As mentioned above, metrics should be defined and used consistently within all activities using that metric. Appendix E provides an example of wording for use in a contractual document to effect a metrics program. This example should be tailored to each SEE. Other useful metric information can be found in the SEI report, *Software Measures and the Capability Maturity Model* (Baumert, 1992) in which 13 different measurement categories are discussed in detail.

3.1.3.7 Data Management and Analysis

In managing today's complex system and software development efforts, the Program Manager has to sort through a huge volume of information and data from a variety of sources to establish project status. Frequently, such information and data (e.g., financial data, personnel accounting data, problem report status, software development metrics, software project data) are stored in databases. To ensure that these items can be integrated into a project-wide picture for management analysis, the underlying formats must have interface consistency. Program Managers can ensure a measure of database consistency by emphasizing the use of standard data dictionaries and data formats.

3.1.3.8 Life-Cycle Maintainability

Program Managers should use an in-house organization to monitor the software development regardless of whether or not the developers are in-house. The in-house organization participates in software technical reviews, performs milestone Independent Verification and Validation (IV&V) activities, and participates in formal testing. If the in-house organization is the future SSA, the project can be guided to meet its requirements for maintainability. This will simplify the transition of responsibilities and lower the long-term costs.

Traditionally, 70% to 80% of a software project's costs are associated with software maintenance during PDSS. Applications developed in Ada using sound software engineering are expected to significantly reduce this cost as a result of the following:

- Fewer defects per line of code because of Ada's strong typing, packages, and exception handling
- Easier understanding of the code for both defect correction and enhancements
- Increased portability of Ada source code to newer hardware if required later in the life cycle
- Shared cost of defect maintenance and enhancements for Ada code reused by other organizations.

For the PDSS portion of the life cycle, the prime developer, another developer, or in-house DON personnel may perform the work. Using the system developer to perform the PDSS avoids incompatibility problems between the development and PDSS organizations. The developer knows and understands the software better than anyone else, hence can maintain it effectively. The major disadvantage is that the Government can become locked into perpetual support performed only by the system developer and, therefore, may be unable to control the costs. The selection of an Ada environment for the project will significantly reduce the problems traditionally associated with the transition from software development to PDSS in the event that the system developer does not perform PDSS.

The transition to PDSS may also require undue effort if the development and PDSS environments are not the same or are incompatible. One alternative, which is often costly, is to procure an identical support environment and install it at the PDSS site. Regardless of the option chosen, developers must prepare for the complete life cycle before and during development. In part, this may be accomplished by requiring delivery of a complete, high-quality package, including tools and documentation sufficient to understand, regenerate, and maintain the software products, as well as delivery of the software.

The Program Office should ensure that test and analysis tools acquired for the sustaining engineering effort allow life-cycle support staff to work with the latest proven technology. Commercial Off-The-Shelf (COTS) tools in the Open Systems Environment (OSE) are becoming more capable of supporting Ada applications. Tools are emerging to support reverse engineering, to import Ada source code into

higher-level design methods, and to assist in the conversion of code from other High Order Languages (HOLs) to Ada.

The ISEA and LCSA efforts require the same engineering disciplines that are applied to the initial development of the system. Life-cycle support consists of a repetitive cycle of development projects. The common denominator is that each cycle builds on the results of the previous one. In the past, after a few revision cycles, COBOL, FORTRAN, JOVIAL, and Compiler Monitor System (CMS-2) implementations typically resulted in significant degradation of the supporting design documentation, the quality of embedded comments, and the integrity of the system architecture.

The combination of well-defined programming standards and Ada's syntactic support for modularity, abstraction, localization, and uniformity alleviates this problem by providing the capability to quickly isolate and correct faults. In addition, automated documentation tools can reduce cost and provide a vehicle for timely updates. The software engineering goals described in Section 5.1.1 are the cornerstones for ensuring long-term, cost-effective maintainability and ease of revision of the supported operational program software written in Ada. Although the use of Ada does not guarantee ease of maintenance, the use of Ada with software engineering is expected to provide significant cost reduction during the PDSS.

3.1.3.9 Information Sources

Information on the political, regulatory, commercial, and technological factors crucial to successful program planning and execution is readily accessible in a variety of forms and formats. This guide is a primary source of information for the DON Program Manager, and it references many of the most frequently used sources of information on implementation guidance. The appendixes provide additional helpful sources for general information, research tools, program assistance, and information technology products and services.

3.1.3.9.1 Resource Organizations

The many organizations listed in Appendix A, Section A.1.1, can be invaluable in providing practical information as well as the more theoretical information surrounding the use of Ada and software engineering principles. Both types of information are useful in program planning.

3.1.3.9.2 Repositories

In addition to the conventional information sources, the Program Managers have many traditional and newer electronic repositories of information and development assets at their disposal, including the following:

- Legacy software in relevant application domains that may provide the functionality required and prevent "reinvention of the wheel."

- Software assets (Ada components) contained in DOD and DON reuse libraries that can provide software-engineered components for inclusion in new developments.

- Information clearinghouses, such as the Ada Joint Program Office's (AJPO's) Ada Information Clearinghouse (AdaIC), which contains the latest information on the state of the practice and project experiences. Appendix A, Section A.2, has information on AdaIC.

- Databases containing program requirements data.

- Benchmarks (best practices) of industry and Government that may be used as program resource requirements, especially in the early planning stages.

- Lessons learned from other organizations.

3.2 ACQUISITION PLANNING

Section 2 provides policy guidance. Documents used to ensure that the software development organization selected is competent in Ada and has the requisite systems and software engineering experience and to identify what and how the developer is to deliver include the following:

- Acquisition Plan
- Statement of Work (SOW) (part of RFP)
- Proposal preparation instructions (part of RFP)
- Proposal evaluation criteria (part of RFP)
- Government estimate
- Deliverables—Contract Data Requirements List (CDRL).

3.2.1 Acquisition Plan

Acquisition planning is critical to the success of any program. From the software perspective, the following requirements should be included in the Acquisition Plan:

- Use of Ada

- Application of software engineering principles

- SLOC definition and estimate as discussed in Section 3.1.3.6

- Software reuse as discussed in Section 3.3.7
- Criteria for establishing and using metrics
- Criteria for evaluating a contractors's Ada development capabilities
- Criteria for evaluating the contractor's software engineering process (Software Developer Maturity Level [SPA or SCE])
- Criteria for evaluating the contractor's risk management capabilities.

3.2.2 Statement of Work

Use of Ada should be specified throughout the SOW. The SOW should clearly identify those areas that will not be considered for an Ada waiver and those that may be subject to an Ada waiver, based on further analysis. In addition, the elements subject to Ada reuse from the Government-Furnished Equipment (GFE) and other domains should be specified. Other issues to be addressed include the following:

- Project application of SLOC definitions
- Definition of all categories of software and firmware
- Reserve capacities and software test acceptance criteria
- Estimated software sizes by configuration items
- Tailoring of applicable standards (e.g., DOD-STD-2167A and DOD-STD-7935A) (See Section 2 for additional policy guidance.)
- Software metric requirements (see Appendix E)
- Requirements of Development Plan, Configuration Management Plan, Quality Assurance Plan, and Life-Cycle Management Plan
- Integrated automated documentation
- Use and delivery of test software, support software, and project-specific data files.

Appendix E lists other contract considerations.

3.2.3 Proposal Preparation Instructions

Depending on the scope of the project, it is often beneficial to have the contractor submit draft planning documents as part of the RFP package and respond to the following issues:

- Commitment to Ada-oriented system and software engineering processes
- Demonstrated corporate adoption of Ada (e.g., Ada training, Ada experience)
- Procedures to coordinate with the Federal Acquisition Regulations (FAR) for reuse of a COTS and/or Government Off-The-Shelf (GOTS) assets
- Statements regarding rights and data issue (i.e., data rights escrows, documentation, quality and testing, software liabilities, contract incentives) with respect to proprietary Ada technologies
- Procedures for enforcing the tenets of Ada and software engineering in subcontractor environments
- Software development process control
- Software metrics program
- Performance of initial allocation of function to configuration items with specification of language and estimated SLOC
- Summary of successful performance on previous programs requiring the use of Ada.

3.2.4 Proposal Evaluation Criteria

The RFP evaluation criteria should include the following:

- Requirements traceability
- Cost modeling
- Ada experience
- Plan for acquiring the required Ada-capable staff needed to meet contract requirements
- Ada reuse assets

- Ada training program
- Ada projects completed and size (e.g., SLOC)
- Adoption of Ada for internal project use
- Proof of an Ada software engineering process and programming culture
- Software process maturity level of repeatable or higher
- Controls and procedures that enforce subcontractor compliance with the proposed evaluation criteria
- Validation that software sizing, cost, and schedule are within Government estimates
- Tools and environments applications
- Testing process
- Risk assessment
- Risk management
- Configuration Management Plan
- Quality Assurance Plan
- SEPG.

3.2.5 Government Estimate

In developing the Government estimate, the following issues should be considered:

- *Experience of Developers*—Developers who have not completed two or three Ada projects will be less productive than more experienced developers.
- *Cost of Code*—Developing code to be reused on other projects is more expensive.
- *Integration of Computer-Aided Software Engineering (CASE) into Software Development Process*—To achieve maximum benefit, CASE tools must be integrated into the software development process.

- *Worst-Case Scenario*—Labor-hour totals and category allocations should be estimated for a worst-case scenario.

- *Product Quality*—Reuse of existing Ada code may lower project costs and result in a higher-quality product.

- *Open Systems Standards*—Use of open standards (e.g., XWindows, Motif, Portable Operating System Interface for Computer Systems [POSIX], Government Open Systems Interconnection Profile [GOSIP]) can significantly reduce cost through improved productivity. A framework of standards for an open systems environment is discussed in Section 5.3.6.

3.2.6 Deliverables—Contract Data Requirements List

The CDRL specifies the deliverables that will be required and the expected quality. Close coordination in preparing the SOW and the CDRLs should result in high-quality software deliverables. The CDRL enables the Program Manager to describe, tailor, and specify, by means of the Data Item Descriptions (DIDs), the nature, detail, and "personality" of the software deliverables. MIL-HDBK-287 provides guidance for tailoring DIDs. Additionally, the National Technical Information Service (NTIS) can provide a PC-based tool called Tailor DID. Appendix A, Section A.1.3, contains information on NTIS.

A detailed CDRL does carry some economic impact. Product quality, however, reduces the life-cycle cost of ownership. Bidders should be instructed to assess the CDRL specifications carefully, and evaluators should be extremely critical of these issues. The CDRL must contain the requirements needed to ensure good systems engineering if the contract is expected to yield a quality product.

The Program Manager should specify that contract data deliverables are to be created and maintained electronically in accordance with the Computer-aided Acquisition and Logistics Support (CALS) Program.

3.3 SYSTEMS ENGINEERING AND RISK MANAGEMENT

Systems engineering should take a programmatic view that recognizes software costs as a significant portion of the total system life-cycle cost. Consequently, systems engineering must be considered in the early stages of a project. During the System Definition Phase, the Program Manager should perform analyses to identify high-risk requirements and potential solutions in the context of both hardware and software. High-risk components should remain visible throughout the development process. Areas of risk management for Ada implementation include the following:

- Software versus hardware in a system context
- Prototyping
- Project context benchmarks
- Requirements volatility and traceability
- Support software acquisition impacts
- Coding for quality
- Ada software reuse
- Prime developer-subdeveloper relationships
- Incremental development
- Integration philosophy
- Testing philosophy, evaluation, and methodology.

3.3.1 Software Versus Hardware in a System Context

A goal of every Program Manager should be to position the acquired system for ease of future support. With software consuming an increasing percentage of total program budgets, the use of Ada software engineering is an important factor in enabling the future support of a system. As for hardware, postponing hardware selection until later in the development cycle can significantly reduce life-cycle costs while providing greater functionality. An excellent case in point is the Federal Aviation Administration's (FAA's) Advanced Automation System (AAS). The AAS contract was awarded in 1988, and much of the software was designed, coded, and tested in Ada using the IBM MVS operating system. In late 1991, with about 40% of the software coded, the newly developed RS/6000 was selected as the hardware for the AAS controller console. The Ada code was easily ported to the RS/6000 environment. By postponing the decision on the hardware, FAA will field the AAS with hardware that is faster, more capable, more reliable, and cheaper over the expected life cycle.

3.3.2 Prototyping

Prototyping is recommended as a method of risk abatement to evaluate human-computer interfaces and alternative algorithms and to confirm requirements, not as a means for rapid deployment. Fielding a prototype system can have life-cycle cost impacts that far exceed the theoretical development cost savings. Prototyping may take the form of a repetitive spiral of alternative solutions that gradually narrows the choices to single out a preferred approach. Still, each alternative must be planned for and accompanied by disciplined documentation of the design approach and system interfaces. Ada directly supports prototyping through the use of separately compiled components.

3.3.3 Project Context Benchmarks

Development of benchmark programs that represent the critical aspects of a software application (e.g., Kalman filters, operating system overhead, correlation algorithms,

graphic output) is recommended to determine whether available compilers and/or code generators satisfy projected operational requirements. Benchmarks also help to quickly identify hardware deficiencies in the project. Section 4.6.1.3 provides further discussion of benchmarks.

3.3.4 Requirements Volatility and Traceability

Requirements volatility increases costs, schedule, and undetected error rates. To gain the maximum productivity from Ada, the software requirements should be baselined at the Software Requirements Review (SRR), but certainly no later than at the Preliminary Design Review (PDR). At PDR, the allocation of software functions to Computer Software Configuration Items (CSCIs) should be complete. At Critical Design Review (CDR), allocations of these functions to respective Ada package specifications should be complete. Any change in requirements beyond this point will affect the cost and schedule. Redesigns required by requirement modifications have the side effect of invalidating baseline documentation; thus, the effort required to change the affected documents must be considered. Prototyping, configuration management, change control, and design revision can reduce requirements volatility. The configuration management method should support traceability of the requirements through design, coding, and testing as well as from testing back to requirements.

3.3.5 Support Software Acquisition Impacts

When using COTS software, the Program Manager must consider the stability of the product vendor to ensure the support environment will be available for the life of the system. The identification of data and data rights must be part of the acquisition of any COTS software. Appendix M lists several publications on data rights that are available from either NTIS or SEI.

3.3.6 Coding for Quality

The AJPO has suggested the use of the *Ada Quality and Style Guide* for the development of high-quality Ada applications. The Software Productivity Consortium (SPC) wrote this guide in 1989. Since then, it has become the standard guidebook for Ada programming at many organizations. The AJPO has negotiated with SPC to use, copy, modify, and distribute this guide for any purpose and without a fee provided it is distributed with the copyright notice. The guide is available in hardcopy through the Defense Technical Information Center (DTIC) and the NTIS. Electronic copies in both ASCII and Postscript are available electronically on the *ajpo* host and on the AdaIC Bulletin Board. Appendix A, Section A.1.3, provides information on DTIC and NTIS.

3.3.7 Ada Software Reuse

Software reuse is important to DOD and the Ada community because reuse promises a higher-quality product with significant cost savings across the software life cycle. Although software reuse was not specifically identified as a design criterion for the Ada language, the concept of reuse is strongly implied in the design criteria.

Domain analysis is a method of identifying the commonalties and variations of existing systems within the same application area to determine reuse candidates. Domain analysis should be performed in the Requirements Analysis Phase to gain a better understanding of existing systems in terms of common functionality and to identify components from existing systems that can be reused in the current development. A side benefit of domain analysis is stabilizing the requirements through the early discovery of missing requirements.

Use of Ada as the implementation language does not, by itself, ensure reusability. Among other things, a library of reusable programs and established Ada programming standards, policies, and procedures are necessary. Libraries of reusable Ada programs have great promise for reducing future software development costs. The Program Manager must be aware of issues that will make maintenance of the library attractive to the development contractor (e.g., data rights). Section 5.3.3 provides additional details on software reusability, and Appendix A, Section A.1.5, provides a list of software repositories.

3.3.8 Prime Developer-Subdeveloper Relationships

In selecting subdevelopers, the prime developer should apply the Ada technology acquisition criteria discussed throughout Section 3.2. The prime developer must be as analytical in selecting its subdevelopers as the Government is in selecting the prime developer. For its part, the Government should consider the subdeveloper's capabilities and commitment to Ada in the same light as those of the prime developer. The relationship between the prime developer and its subdeveloper should not be taken for granted. The RFP should specify that the developer-subdeveloper relationship be explicitly described in the proposal and should include proposal evaluation criteria that address this relationship.

3.3.9 Incremental Development

Shrinking budgets as well as poorly defined and shifting requirements have put many programs at risk. To reduce risk, one strategy employed by Program Managers is to address software developments in increments. This facilitates the reevaluation and optimization of requirements when moving from one incremental phase to the next. The software engineering features of Ada (e.g., packages, generics, information hiding, and abstraction) support this risk-reducing approach.

3.3.10 Integration Philosophy

The size and complexity of DON systems are increasing. As a result, systems integration is growing in proportion and importance. Both the tactical and nontactical worlds are developing highly complex, multifunctional systems, and the distinctions that once existed between the systems used in these two worlds are becoming blurred. A classic example is the critical time requirements of modern logistics systems to support the rapid deployment and multimission functionality of today's naval forces.

Ada use increases in importance in light of the diversity of hardware required for the wide variety of these operational environments. Ada can be applied over most of these environments for mission performance and problem solving. With the appropriate systems engineering practices and policies, Ada could contribute significantly to easing the multivendor, multimission integration problems encountered in today's systems.

3.3.11 Testing Philosophy, Evaluation, and Methodology

Because of the increasing complexity and size of emerging systems, a testing philosophy must be considered and built in at the concept formulation phase of a project. All requirements, developed and derived, must be evaluated not only for their application to the problem statement or mission, but also for testability. Large-scale, heterogeneous, distributed systems must be thoroughly tested as modules and subsystems as the system is built and comprehensively tested as an integrated system. The appropriate use of Ada components should support the testing philosophy by providing a capability to confirm the implementation of requirements and evaluate them for completeness. Because the components provide a defined external interface, integration and testing problems are minimized. Test planning must start in the Requirements Definition Phase (e.g., requirements validation, requirements realism, and transitioning of requirements to function). Contracts must ensure and enforce requirements for unit testing of the individual functional capabilities; full-scale integration testing; verification, validation, and certification testing; and life-cycle support testing.

Ada supports the identification of errors during the early Design and Coding Phases through compilable program specifications, strong typing, and constraint checking. Constraint checking takes place statically during compilation and dynamically during run time where an exception is raised and can be easily processed. This constraint checking can be disabled to improve system performance after all testing has been completed. These features help to uncover errors earlier in the development process. In the end, the cost impact of error repair during integration testing demonstrates that this phase should focus on integration, not discovery of unit code errors, bad design, or ambiguous requirements.

In the past, the error correction process often applied patches to the source code because complete system generation was time consuming. Patches are unnecessary when the system is properly designed by using Ada packages. Errors corrected within a package body require only the recompilation of the package body and relinking, which significantly reduces the time for a complete system regeneration.

Testing methodologies include the use of designed-in test structures and the application of tools for modeling, simulation, design, development, and operational assessment of systems in both the hardware and software arenas throughout the development process. When used as part of a well-defined systems engineering approach, Ada supports an incremental build and test approach.

3.4 HIGHLIGHTS

The preceding sections provide general guidance for Program Managers in incorporating Ada and software engineering into a program's life cycle. This subsection highlights the critical points in each phase.

Guidance for the Program Planning Phase, for example, emphasizes the need for integration, up-front preparation of procedures, and careful tool analysis and selection. This guidance may be summarized as follows:

- Ensure the program organizational structure(s) integrates the activities of the Program Office, the operational users, and developers; embodies clearly defined lines of authority; and supports promotion of Ada use and enforcement of Ada policy and standards.

- Ensure that Ada expertise is available within the management and support organizations.

- Use parametric modeling and cost-estimating tools. Ada sizing inputs to estimation models should be based on Ada statements or terminal semicolons rather than on lines of code.

- Develop a training strategy:
  - Address software engineering and Ada training.
  - Ensure appropriate training at all levels of the organization.
  - Ensure the training is just in time.
  - Use on-site mentors.

- Identify, document, and disseminate standards and procedures needed to manage the software reuse and development process.

- Identify metrics to be applied at the beginning of the systems engineering and software implementation processes.

- Take advantage of the information sources available on political, regulatory, commercial, and technology issues, and lessons learned from other projects.

Guidance for the Acquisition Planning Phase relates primarily to the requirements of the RFP. Highlights of this guidance are as follows:

- Develop a strong acquisition plan and other documents needed to ensure selection of a competent Ada developer.

- Ensure the RFP SOW provides specifications on Ada use and/or reuse, defines all of the applicable software and firmware categories, specifies conformance to applicable standards, and states the software metrics requirements.

- In the RFP evaluation criteria, emphasize contractor experience, training, and competence in Ada use and reuse as well as requirements traceability, cost modeling, and subcontractor compliance with the evaluation criteria.

For the Systems Engineering and Risk Management Phase, the guidance provided stresses the importance of considering software-related issues from the outset and identifying high-risk hardware and software components. Specific guidance in this area includes the following:

- Define software requirements in parallel with hardware requirements.

- Use prototyping and benchmarks to mitigate risk.

- Freeze requirements no later than the PDR.

- Identify opportunities for and consider reuse of Ada source code.

- Ensure the prime developer applies the same Ada technology acquisition criteria in selecting subdevelopers as the Government used in the selection of the prime developer.

- Implement processes that identify errors early in the Development Phase to avoid costly repair during the Integration and Test Phase.

- Ensure that the cycle of incremental builds, test, and repair are under configuration management.

Section 4

Environments

This section discusses the environments used to support the development and maintenance of Ada application software.

4.1 PROJECT SUPPORT ENVIRONMENT

The term "environment" evolved from early work on the UNIX operating system and originally referred to setting certain parameters and characteristics of the operating system to create a programming or computing environment suitable for the user. From the start, the term implied selecting or tailoring the environment to meet specific user needs. Gradually, the term became more general, referring to Programming or Project Support Environments (PSEs), thus reflecting the idea that the environment was designed and constructed to support a particular class of applications. The phrase "Software Engineering Environment" (SEE) was then used to refer to a set of computer tools to support the activities of software engineering.

A SEE that supports software development is a computer-based set of integrated methods, tools, and procedures to develop software that meets mission needs. This definition inherently means that a SEE must provide the functions of both a programming development system and a management information system to monitor and control the development. The term "Ada environment" means that the scope of the SEE is focused on supporting the development of software written in the Ada programming language and the associated software engineering methods and procedures.

Currently, the accepted practice is to execute the SEE on host computers to develop project software during the Requirements, Design, Coding, Integration, and Testing Phases. Normally, system integration and testing are performed in system test facilities with support from the SEE. The SEE is then used for the remainder of the system life cycle.

Other terms for SEEs that appear in the general literature include Software Development Environment (SDE), Integrated Software Engineering Environment (ISEE), PSE, Ada Programming Support Environment (Ada PSE), and Integrated Project Support Environment (IPSE). The term PSE, which appears in the remainder of this section, is used interchangeably to mean project and programming support environments.

4.2 TOOLS

A variety of tools is used in PSEs. These tools may vary significantly from one PSE to another. A few tools are used in almost every application although several other tools exist that are desirable and can improve productivity across the software life cycle.

4.2.1 Minimum Tool Set

The minimum tool set consists primarily of those tools associated with the coding phase of development, including the following:

- *Editor* is used to create or modify source code and to view or modify files produced by other tools.
- *Compiler* translates a High Order Language (HOL) source program into its relocatable code equivalent.
- *Assembler* translates an Assembly source program into relocatable code.
- *Linker* creates a load module from one or more independently translated modules by resolving the cross-references among the modules. Some vendors separate part of this functionality and provide it in a separate binder tool.
- *Relocating Loader* executes on the host computer and translates the relative addresses into the absolute addresses and produces an execution module.
- *Run-Time Environment (RTE)* provides a variety of operating-system-like services for application programs (also known as run-time executive).
- *Profiler* provides a mechanism to monitor the dynamic aspects of an application (e.g., scheduling, Central Processing Unit [CPU] utilization, Input/Output [I/O] channel loading).
- *Simulator/Emulator* simulates or emulates the target computer but executes on the host computer and greatly increases testing productivity.
- *In-circuit Emulator* emulates, tests, and traces the prototype system operation when connected to the prototype system through the microprocessor socket.
- *Symbolic Debugger* allows a programmer to test a module by controlling the program execution on a target computer emulator or the target computer itself by using source program symbols or names.

- *Pretty Printer* automatically applies standard rules for formatting program source code.

- *Host-to-Target Exporter* provides a tool to transmit the execution module from the host to the target when the target machine is different from the host machine.

A more comprehensive set would include tools covering the following categories: syntax-directed program editing, configuration management, system modeling, Ada program design, software specification and design, software technical documentation aids that facilitate DOD-STD-2167A document production, software quality and performance analysis, and project management. Such tools are often referred to as Computer-Aided Software Engineering (CASE) tools. Appendix F provides a more detailed discussion of these tools.

4.2.2 Commercial Ada Development Tools

Ada has matured significantly over the last few years. Today, many vendors supply several hundred validated Ada compilers for a number of commercial configurations. The Ada Joint Program Office (AJPO) updates the list of validated Ada compilers monthly and makes the list available on the AJPO host computer on the Defense Data Network (DDN). To obtain the list, contact the Ada Information Clearinghouse (AdaIC). Appendix A, Section A.2, provides information on AdaIC and the material it supplies.

In addition, hundreds of commercially available tools exist that are backed by their vendors to support the development of Ada applications across the software life cycle. Tools are available to support both hierarchical and object-oriented design. Other tools are available that produce and use Booch diagrams, Buhr diagrams, Demarco data flows, entity relationships, flowcharts, functional flow diagrams, state transition diagrams, and structure charts. Some of these tools provide automatic code generation from the design diagrams.

In addition, tools exist to support source code translation to Ada from Assembly, C, COBOL, Program Design Language (PDL), JOVIAL, LISP, and Pascal. Even Ada artificial intelligence tools are commercially available to support the building of expert systems, knowledge-based systems, natural language systems, and neural networks.

The AJPO maintains an on-line Ada Products and Tools Database that can be used to find out more about these tools. In addition, the Software Technology Support Center (STSC) maintains information in the form of a database and reports for tools

to support a SEE. Appendix A, Section A.1, provides more information on both AJPO and STSC.

4.3 MISSION-CRITICAL COMPUTER RESOURCES ENVIRONMENT

Department of Navy (DON) software applications for mission critical systems run on a wide variety of target computer systems including commercially based microprocessors and computers and military-unique computers such as the Navy Standard Embedded Computer Resources (SECR). Often the program support environment tools selected for such applications run on a host computer system that is different from the target computer system. The host system provides the resources for development, simulation, documentation, and test of software applications to be compiled for the target.

Navy SECR computers include the AN/UYK-43(V), AN/UYK-44(V), and the AN/AYK-14(V). The Ada Language System/Navy (ALS/N) serves as the validated Ada compiler, the run-time software, and the PSE for the SECR computers. Section 4.5.4 provides further discussion of ALS/N. For Navy commodity-managed computers, such as the Desktop Tactical Computer (DTC-2) and the Tactical Advanced Computer (TAC-3), validated Ada compilers may be selected through the DTC-2 and TAC-3 ordering contracts for the commercial derivatives of the DTC-2 (Sun SPARC Workstation) and TAC-3 (Hewlett-Packard [HP] Workstation 9000) Because the DTC-3 and TAC-3 are based on widely used commercial workstations, a wide variety of commercial PSEs are available to support Ada software developments.

Many DON mission-critical Ada applications have been developed and fielded successfully by using commercial processors and/or computers, validated commercial compilers, and commercial PSEs. Use of commercial computer resources, compilers, and PSEs has greatly facilitated and accelerated Ada implementation within the DON.

4.4 AUTOMATED INFORMATION SYSTEMS ENVIRONMENT

The Congressional mandate to use Ada throughout the DON will result in having many Automated Information System (AIS) applications programmed entirely in Ada. The AIS community uses a wide range of commercial hardware and software. Therefore, the DON strategy for establishing the Ada environment for AIS will incorporate the use of Ada into an Open Systems Environment (OSE) that will provide the capability to integrate and transport application software across multiple computer systems. Current policy requires that, once the Integrated Computer-Aided Software Engineering (I-CASE) contract has been awarded, all tools for AIS environments should be selected from this contract.

4.5 PROJECT SUPPORT ENVIRONMENT OPTIONS

PSEs are critical to the successful development and maintenance of DON computer systems. Ada PSEs include commercial Ada environments, ALS/N, AdaSAGE, and Ada-Based Environment for Test (ABET).

4.5.1 Commercial Ada Environments

The commercial Ada PSEs based on validated Ada compilers are steadily increasing in number and becoming more mature. Commercial Ada PSEs typically contain a set of system tools that provides capability for data management, resource management, and scheduling. Additionally, they provide the target RTE with loaders, debuggers, and the like. Although the completeness and quality of these Ada PSEs vary, several highly capable Ada PSEs have evolved.

These commercial environments have been exceptionally powerful for the development of Ada software on both host and target environments. A wide range of CASE tools, which are discussed in Appendix F, support these commercial Ada environments. The AdaIC and STSC, which are described in Appendix A, Section A.1.1, also can provide information on these environments.

4.5.2 AdaSAGE

AdaSAGE is an applications development set of utilities designed to facilitate rapid construction of systems in Ada. Applications may vary from small to large multiprogramming systems that use special capabilities. These capabilities include Structured Query Language (SQL)-compliant database storage and retrieval, graphics, communications, formatted windows, on-line help, sorting, editing, and more. AdaSAGE operates on MS-DOS and UNIX platforms, and AdaSAGE applications can be run in the stand-alone mode or in a multiuser environment. A developer using the Ada language and the AdaSAGE development system can design a product that is tailored to a specific requirement and offers outstanding performance and flexibility.

AdaSAGE is an effective environment for developing software applications primarily for the AIS and some scientific and engineering domains. The environment allows the user to build applications through an interactive screen editor. Here, the applications developer is presented with options that are based on reusable modules in the AdaSAGE program library. After all options have been selected for an application, the AdaSAGE environment builds the Ada source code, which then can be compiled to create the application. Developers with limited knowledge of Ada or of software engineering can develop simple applications. Development of larger applications may require programmers skilled in Ada and software engineering. Functionality and potential benefits of AdaSAGE include rapid prototyping, programmer reusability, and efficiency.

All four Services have reported significant success in developing applications with AdaSAGE. The Department of Energy and private industry also are using AdaSAGE.

Additional enhancements to this environment are under way, funded by several sources including the AJPO's Ada Technology Insertion Program (ATIP).

4.5.3 Ada-Based Environment for Testing

ABET is a set of software interface standards for Automated Test Equipment (ATE) environments used to assist in the development and execution of automatic tests using the Ada programming language. These software interface standards are defined to support hardware or software component portability, reusability, exchangeability, and interoperability and to serve as targets for test-related software development tools.

Each ABET interface is defined in one of the Institute of Electrical and Electronics Engineers (IEEE) ABET component standards (IEEE Std 1226.x). The ABET component standards define the Ada packages and data structures that are used to give a comparable capability to the associated reference documents. They also define mappings between the reference documents and the Ada packages and data structures.

The ABET interface standards can be grouped in an ABET layer model. The layers of this model are as follows:

- The *Product Description Layer* supports the link from design-oriented product description information to test-oriented product information. The design data may include descriptions of the Units Under Test (UUTs) physical and circuit design and its externally measurable characteristics and responses and the stimuli needed to elicit them. Standardization of these interfaces supports analysis of test and maintenance issues during the Product Design Phase as well as development of automated tools to generate the requirements or procedures directly from UUT product descriptions.

- The *Test Strategy/Requirements Layer* supports the specification of UUT test requirements, test strategies, automatic test generation, diagnostic models, and collection and retrieval of maintenance data.

- The *Test Procedure Layer* supports full signal-oriented vocabulary and semantics and the UUT-oriented virtual resource model of ATLAS and includes the capability to relate UUT test requirements to virtual test resources that are independent of any particular ATE.

- The *ATE System Layer* specifies the mapping of virtual resources to real resources, signal routing, and access to non-UUT signal-oriented test functions and maintains and reports on ATE status.

- The *Instrument Control Layer* standardizes access to command and communication protocols used by real test resources. This will allow test resources with common functionality made by any manufacturer to be used to perform identical functions.

4.5.4 Ada Language System/Navy

ALS/N is a software development and RTE that is being developed for the current generation of DON standard computers: the AN/UYK-43(V), AN/UYK-44(V), and AN/AYK-14(V). The ALS/N development is scheduled to be completed by the end of Fiscal Year 1993. Present plans call for two additional years of maintenance; after that period, user projects will be required to provide support for maintenance.

ALS/N, which is hosted on the VAX series of computers using the VMS operating system, has been validated as Ada/L for the AN/UYK-43(V) and as Ada/M for the AN/UYK-44(V) and AN/AYK-14(V).

ALS/N consists of two functional parts: the Minimal Ada Programming Support Environment (MAPSE) and the RTE. The MAPSE consists of the compiler and other associated compile-time tools that run on a host computer and produce software products for a target computer.

The RTE software provides services needed by the executable application program and supports execution of those programs so as to meet the requirements of performance, reliability, and fault-tolerance on the target computer. The RTE provides the basic and extensible software facilities required to support Ada use in the mission, support, systems, test and maintenance, and trainer software categories. RTE tools include the run-time operating system, executive, librarian, loader, run-time application support, run-time debugger, embedded target debugger, and run-time performance measurement aids. The ALS/N RTE provides run-time support for the AN/UYK-43(V), AN/UYK-44(V), and AN/AYK-14(V) embedded target computers only.

4.6 SELECTION OF THE PROJECT SUPPORT ENVIRONMENT

The subsections below provide PSE-related information for compiler selection, availability of PSE standards, upgrades to newer versions of tools within a PSE, mixing Ada with other languages, and mixing executable Ada from different compilers.

4.6.1 Compiler Selection

A compiler is a software tool or product that receives as input the HOL or source-language statements developed by designers and/or programmers and translates or compiles these statements into machine-readable, executable code. Vendors typically provide a set of tools in addition to the compiler and call the collection a "compilation system." A typical compilation system includes only a compiler, linker, loader, library, and fundamental program execution (run-time) structure. Other vendors package additional tools (e.g., interfaces and support for programmer productivity, testing and configuration management tools, and structures) into the compilation system. Consequently, these packages are almost complete software development environments.

Each manager must select the best compiler for particular project needs. The compiler selection process should begin with a plan that establishes the project requirements, budget, personnel, and timetable. To reduce risk, the selection process must identify key criteria and test the candidate compilation systems against the criteria. The criteria for selecting a compiler should be based on the nature of the project; for example, concern about execution time would not be as applicable in a batch-type application as in a tactical program. Based on these criteria, benchmarks, checklists, and interviews should be used as needed to assess different compilers for specific project requirements, and a limited number of candidate compilers should be selected for detailed evaluation. Several types of benchmarks and test suites can be used to evaluate compiler implementation. No single test suite or checklist suffices for every project (Weiderman, 1989).

Evaluation of Ada compilation systems for a particular user application will decrease project risk and reduce total cost and schedule overruns. Evaluation and selection apply to the entire software development package, not just the compiler. Evaluating and selecting an Ada compilation system for a project are complex and costly processes. The means available to support them, as discussed below, are very important, but it will always be necessary to supplement them with project-specific criteria.

4.6.1.1 Validation

All Ada compilers must pass formal validation to ensure conformance to American National Standards Institute (ANSI)/Military Standard (MIL-STD)-1815A. DOD Directive 5000.1 and DOD Instruction 5000.2 require that all DOD initiatives use validated compilers. A list of validated compilers can be obtained from the AdaIC. The formal validation consists of approximately 4,000 tests known as the Ada Compiler Validation Capability (ACVC). A formal validation ensures that an Ada compiler correctly implements the Ada language syntax as defined by the standard. A validation does not, however, assess performance or machine-dependent language

features. Section 4.6.1.3 provides a discussion of benchmarks. Ada compiler developers who want to validate their products may obtain information on the current version of the ACVC test suite from the AJPO or the National Institute of Standards and Technology (NIST).

Project Managers must select a validated Ada compiler for their software development projects. Validations are issued for a tested host and target combination with an expiration date. It is important to verify that selected compilers have been validated against the most recent version of the test suite. The timing of compiler procurement should correspond with the start of the project. A validated compiler used at project start is considered validated for the entire life cycle of the designated project, even if the expiration date has passed.

4.6.1.2 Evaluation

The goal of formal evaluation is to provide vendors, procurers, and users of Ada implementations with comparable compiler performance data. These data enable vendors to:

- Improve compiler implementation performance
- Allow procurers to select implementations and configurations that best meet their project needs
- Help users identify the best language features to use for that particular application
- Help users identify the language features to avoid for that particular application.

Two systems are available for Ada compiler evaluations: the Ada Compiler Evaluation Capability (ACEC) and the Ada Evaluation System (AES). Work is under way to merge the ACEC and AES into the Ada Compiler Evaluation System (ACES). The merged product is expected to be available in mid-1993.

4.6.1.2.1 Ada Compiler Evaluation Capability

In 1983, AJPO formed the Evaluation and Validation Team to examine PSE and tool evaluation and validation issues and provide a capability to assess Ada PSEs to determine their conformance to applicable standards. One result was the ACEC. The current ACEC is available from the Data and Analysis Center for Software (DACS); information on DACS is available in Appendix A, Section A.1.5.

The ACEC strengths include the depth of coverage for language features, documentation, documented structure, code size measurements, timing techniques with a statistical model, and cross-system analysis software. The shortcomings are lack of support; limitation of an automated analysis subsystem; weakness in testing compile-time performance; and lack of diagnostics, debuggers, and a library system.

4.6.1.2.2 Ada Evaluation System

The AES is a test suite designed for the British government to perform testing of an Ada programming environment. Measurements are taken on features such as compile- and execution-time performance, generated-code quality, compiler-produced error and warning messages, linker and library systems, and debugging capabilities. AES strengths are breadth of coverage, interactive user interface, automatic generation of reports, extensive documentation, macro capability for test generation, checklist for diagnostics, library systems, vendor evaluation, and examples. AES shortcomings are considerable setup time, lack of U.S. support, cost, target run-time performance, and the subjective nature of the checklist.

4.6.1.3 Benchmarks

All compilers are not alike. Benchmarks provide techniques and application examples to compare performance among different Ada compilers or performance among Ada compilers and other language compilers. Careful analysis of the benchmark result will help identify the compiler best suited for the computing environment of a given project. To guarantee successful benchmarking, the benchmarks best suited to project requirements must be selected or developed. For some medium and large projects, it may be necessary to develop benchmarks that reflect specific project needs. Program Managers cannot depend solely on publicly available resources; they will have to generate some of their own benchmarks.

When selecting and developing benchmarks, the Program Manager should ensure the following:

- Benchmarks adequately represent and test the system requirements and the environment selected.
- Benchmarks are part of a planned, total, integrated, supported test suite.
- Benchmarks are maintained throughout the total life cycle.
- Benchmarks and benchmark requirements are suitable for inclusion in a contract.

The benchmark developed by the Performance Issues Working Group (PIWG) of the Association for Computing Machinery (ACM) Special Interest Group on Ada (SIGAda) comprises a suite of Ada performance measurement programs that focuses primarily on the execution time of individual features of the Ada language. Many tests in this suite are designed to be machine independent and to run without modification. These tests fall into the areas of clock resolution, task creation and rendezvous, dynamic storage allocation, exception handling, array processing, procedural and run-time overhead, composite benchmarks, compilation speed, and capacity test. The strengths of the PIWG benchmark are ease of use, wide use, wide distribution, low run time, and no cost (it is free through the DDN). The weaknesses of the PIWG benchmark are lack of documentation and support.

The Software Engineering Institute (SEI) is developing new benchmarks for Ada applications.

4.6.2 Availability of Project Support Environment Standards

Over the past few years, national and international standards bodies have expended a great deal of technical effort to define reference models and interface standards for PSEs, such as those discussed in Section 7.5.1 related to the Project Support Environment Standard Working Group (PSESWG), Section 7.6.2 (i.e., Portable Common Tool Environment [PCTE]), and Section 7.7 (i.e., Portable Common Interface Set [PCIS]). The eventual fruition of these efforts will allow greater modularity, flexibility, data interchange, and openness among tool suites that constitute PSEs. The Navy Next Generation Computer Resources (NGCR) program, described in Section 7.5, has been participating in the definition of standards.

Until these standardization efforts are more mature, the tool suites that constitute PSEs for Ada (and all other programming languages) will be product-driven. Consequently, Program Managers need to carefully manage the risks that may be associated with the following:

- Migrating from one PSE to another
- Identifying the essential elements of the development-phase PSE that must be preserved for the maintenance-phase PSE
- Upgrading various elements of the PSE to incorporate the latest available technology.

A possible interim strategy to minimize these risks is to select PSE tool suites from commercial vendors that have committed to support these emerging national and international PSE standards in their product lines.

4.6.3 Tool Upgrades in a Project Support Environment

During the project life cycle, it may be desirable to upgrade operating systems, compilers, editors, CASE tools, and the like to the latest version. The Program Manager is responsible for controlling such upgrades wisely because even minor upgrades can have serious cost and schedule repercussions.

4.6.4 Mixing Ada With Other Languages

DON policy guidance requires the use of Ada in major software upgrades for computer systems. Choices must be made between redesigning and recoding all of the software in Ada or developing a strategy to support systems with a mixture of Ada and other languages.

Completely redesigning and recoding the system in Ada are usually cost-prohibitive activities although they are necessary to achieve the full benefits of Ada. Simply translating the existing non-Ada code into Ada carries the risk of code expansion and does not take advantage of the many software engineering features of Ada. Cohabitation of Ada code and code in other languages appears to be a necessary option—but it entails risk and requires much corporate planning and commitment.

One approach is to interface existing code to an Ada program by using language interface features. In this case, compilers that support the other languages must be selected. Many compilers have excellent interfaces to languages such as COBOL, FORTRAN, and Pascal.

A second approach is to isolate the unique languages by the processor on which they are to run (i.e., using only one language on a given processor). This approach allows interchange of data through message passing or similar methods of data normalization and is a low-risk approach because it minimizes the need to change the old code while eliminating new code constraints. For this approach, compilation systems for each of the language-processor combinations will be needed. This requirement affects PSE selection only when the desire is to have a single PSE capable of supporting all of the languages in use in the system.

4.6.5 Mixing Executable Ada Programs From Different Compilers

Another class of problems is encountered when a project tries to mix executable Ada programs from two or more commercial compiler systems on one target computer. In general, such programs are incompatible because of the differences in calling sequences and RTE. One approach to this problem is to take advantage of the portability of Ada programs at the source level. In this way, even if two different compilation systems were used to develop the application software, all source code would be compiled on only one system to create the executable software for the target computer. A second approach is to define an interface between the two

portions of the application. In this way, the interface can resolve the incompatibility issues.

4.7 IMPACT ON POST-DEPLOYMENT SOFTWARE SUPPORT

Post-Deployment Software Support (PDSS) activities, although a microcosm of the development activities in scope, generally take much more time than development activities. Therefore, effective PDSS requires good PSE support. Although few activities or functions are unique to PDSS, some receive more or less emphasis during PDSS than during development. During PDSS, for example, generally there is a heavy emphasis on configuration management, version control, and regression testing.

Transfer of the software product from the development to the PDSS environment is another activity that must be examined. If the two PSEs are identical, this task is much easier. Most often, however, the PSEs will be different. Therefore, the compatibility between the software tools and their interfaces becomes a very real issue that must be solved.

Section 5

Ada and Software Engineering

Rear Admiral Robert M. Moore, the Software Executive Official for the Department of the Navy (DON), provided an appropriate introduction for this section in a speech to the Eleventh Annual National Conference on Ada Technology on 18 March 1993. Admiral Moore stated:

> *In the past several decades, computer technology has played an important and increasing role in building systems which maintain our military superiority. However, the software to run these systems is continuously becoming more complex, more expensive, and takes longer to develop. At one time, it was the hardware that supported the mission; today, the hardware is rather generic, capable of supporting any mission. It is the software that provides the real functionality.*
>
> *We recognize the need to improve the process for developing and acquiring software systems. We recognize the importance of software engineering for developing and maintaining our systems. And we recognize Ada as an important, critical technology, necessary to support good software engineering.*
>
> *Certainly, good software engineering is possible without Ada. Using Ada does not guarantee good software engineering, but Ada as a building block for software engineering does provide a real capability to develop high-quality, reliable systems to satisfy our mission in the fleet and to provide a real capability for support and maintenance over the entire system life cycle.*
>
> *Ada and software engineering have been important to the Department of the Navy in the past; Ada and software engineering will become even more important in the future as we address the new challenges necessary to transition our military force from one capable of defeating a superpower to one capable of maintaining peace in an environment of unremitting Third World confrontations.*

This section addresses the concept of software engineering. It identifies its goals and underlying principles and lists several Ada features that are important in supporting software engineering. It also discusses several Ada technology issues resulting from good software engineering. Appendix L and its attachments expand on many of the topics presented in this section.

5.1 SOFTWARE ENGINEERING CONCEPT

Good systems engineering practices provide a framework for good software engineering practices. The Ada language was specifically designed with features to support software engineering.

5.1.1 Software Engineering Goals

The discipline of software engineering has identified four goals, supported by a number of software engineering principles, to help manage the complex task of developing modern software applications. Figure 5-1 presents these goals and principles. The four software engineering goals are reliability, modifiability, understandability, and efficiency, which are defined as follows:

- *Reliability* is associated with the quality of the software and is a critical goal when the cost of failure is high. Reliability issues must be addressed throughout the design process. Reliability can only be built in from the start; it cannot be added at the end.

- *Modifiability* deals with the capability to perform maintenance on or otherwise change the software. A change in requirements and/or design should result in a controlled change in the software. Error correction to the software should be effected as a controlled change to the software.

- *Understandability* is key to the management of complexity. For a system to be understandable, it must reflect our natural view of the world. At a high level, objects and operations map to real-world data and algorithms. At a low level, the software solution is understandable as a result of proper coding style.

- *Efficiency* refers to the use of resources. Time and space resources should be used optimally. This goal is especially important when real-time deadlines must be met to satisfy the application requirements.

5.1.2 Software Engineering Principles

The software engineering principles that support the goals of software engineering are abstraction, classification, completeness, confirmability, encapsulation, information hiding, inheritance, localization, modularity, and uniformity. The definitions of these principles are as follows:

- *Abstraction* allows users to highlight the essential details of a process or its data dependencies and omit the nonessential details. In this manner, the logic of a program solution can be expressed in terminology approximating the problem domain rather than in computer-dependent terms. Abstraction supports code readability and maintenance. The abstraction principle directly supports the

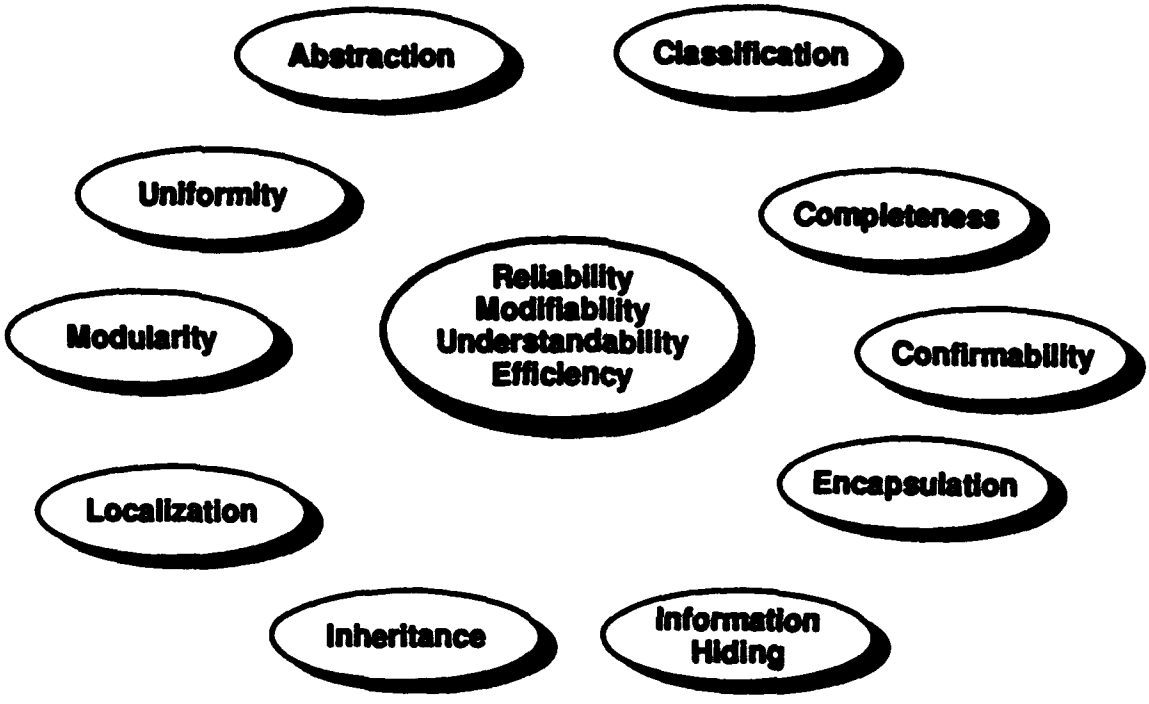


Figure 5-1. Goals and Principles of Software Engineering

goals of understandability and modifiability. The essential details can be filled in later without affecting the balance of the system.

- *Classification* provides a means to comprehend the real world by making generalizations from discrete observations. Classes are organized in a hierarchy with well-defined interfaces. This principle is a cornerstone of object-oriented programming, allowing the creation of specialized objects that inherit the properties and characteristics of an object class. Ada83 supports classification with the use of additional object-oriented tools. Ada 9X has been designed to effectively support object-oriented programming. The principle of classification supports the goals of reliability, modifiability, and understandability.

- *Completeness* creates programs that entirely satisfy both the behavioral and performance software requirements. Completeness helps us develop correct solutions by ensuring that all of the important elements are present. The principle of completeness supports all four software engineering goals.

- *Confirmability* verifies that the application software developed satisfies all requirements. Each software system must be readily testable. Decomposed systems can localize testing and thus help make systems modifiable. Strong typing facilities help the confirmation process. Specialized automated tools that understand the syntax and semantics can also support the confirmation process. The principle of confirmability also supports the four goals of software engineering.

- *Encapsulation* allows users to see only those services that are available from an object. Access to objects is only through well-defined interfaces. Implementation details are hidden from the user. This principle is closely related to abstraction, information hiding, modularity, and localization. The principle of encapsulation supports the goals of reliability, modifiability, and understandability.

- *Information Hiding* makes inaccessible certain details that do not affect other parts of a system. For example, a disk drive should be controlled as a collection of files, but an application should not control a disk drive by using tracks and sectors. Doing so could violate data integrity concepts implemented as part of another process. Reliability of systems is enhanced when, at each level of abstraction, only the necessary operations are permitted, and operations that violate a logical view of that level are prevented.

- *Inheritance* allows the properties or characteristics of an object class to be inherited by a new object. This is an important principle necessary to support object-oriented programming. The principle of classification supports the goals of reliability, modifiability, and understandability.

- *Localization* creates programs in which each part is highly cohesive (i.e., critical data are self-contained) and loosely coupled (i.e., a part can execute in isolation). Localization enables development of self-sufficient components that can be implemented with minimal technical interproject and intraproject communication. Modularity and localization are key components in reducing expensive project communications overhead and critical to incremental build and test. The principle of localization directly supports the goals of modifiability, reliability, and understandability.

- *Modularity* supports the organization of very large programs into discrete parts, which allows separate development of the individual components. The principle of modularity directly supports the goals of modifiability, reliability, and understandability.

- *Uniformity* refers to the use of a consistent notation for all artifacts within a software development activity. Modules are free from any unnecessary differences. A standard coding style used consistently during a project supports the principle of uniformity. The principle of uniformity directly supports the goals of modifiability, reliability, and understandability.

The Ada language was developed to support the software engineering goals through features that draw on software engineering principles. If these principles are understood thoroughly and applied on a project, use of Ada can effectively support the software engineering goals. This facilitates effective systems engineering.

5.2 Ada LANGUAGE FEATURES THAT SUPPORT SOFTWARE ENGINEERING

Ada has several features that directly support software engineering. Appendix L discusses several features many people consider important, including Ada packages, strong typing, exceptions, generics, Ada libraries, and tasking. The programming examples provided illustrate how Ada's special features contribute to increased software quality, performance, portability, and supportability.

5.3 SOFTWARE ENGINEERING TECHNOLOGY PRACTICES

This section addresses major technology practices available today to support the software development and maintenance of today's modern systems. The relevance of these technology practices to Ada is discussed.

5.3.1 Prototyping

Use of prototyping as a standard technology has only recently become widespread. The surge of interest results from the availability of powerful nonprocedural languages; new design and programming techniques; programming languages, such as Ada, that promote use of good software engineering techniques; and the generally recognized cost and reduced-risk benefits of prototyping.

A prototype may be defined as an early running model of a system to be built. This model may represent only a very small portion of the system, such as the interaction among several of the product's computer display screens, so that the evaluation of user-friendliness techniques can be studied. Or, the model may represent a substantial portion of the system so that many of the functions can be demonstrated.

Prototyping can be extremely valuable to the Program Manager in defining and evaluating the system requirements, especially with respect to the user interface. Hands-on experience by system users is the most effective way known of validating requirements, eliminating ambiguities in requirements, identifying requirement deficiencies, and analyzing the design to support the requirements.

5.3.1.1 Purpose

The primary purpose for building a prototype is usually to experiment with, demonstrate, or prove the feasibility of a concept. Prototypes have been valuable in conducting the following activities:

- Evaluating requirements
- Assessing costs of alternative design approaches
- Assessing feasibility of a specific design
- Assessing performance for alternative design
- Determining a product's human-computer interface
- Developing and fine-tuning the product specifications
- Evaluating interactions of parallel threads of an application
- Applying new technology
- Promoting a proposed product to management or to customers
- Obtaining an early start in developing a new product.

Prototyping should focus on assessing and reducing the risks associated with integrating available and emerging technologies into a system design approach to satisfy a validated mission need.

5.3.1.2 Role in Evaluating Requirements

Prototyping the user interface before Milestone II will help identify whether specifications to satisfy the requirements are valid. This determination allows the development activity to correct any deficiencies in the user interface early in the program where changes are less costly. It is often observed that what looks good on paper often does not work in the real world. Consequently, each improvement made to the system specifications early in the product development process can save a significant amount of rework later. Prototyping the user interface has one other very important benefit: a prototype provides an opportunity to learn how the system will behave, in order to further explore the implications of the system requirements.

5.3.1.3 Prototyping Considerations

Generally, all systems that have significant interactions with end users and exceed 5,000 Source Lines of Code (SLOC) are good candidates for prototyping.

Ada is effective in supporting most prototyping activities. The Ada package, which is described in Appendix L, allows software engineers to rapidly develop an architecture for the system by identifying critical interfaces. A stubbing capability in Ada allows code to be separately compiled for ease in developing prototypes. For XWindows environments, Ada Graphical User Interface (GUI) builders are available that allow dynamic creation and evaluation of the user interfaces.

Prototypes intended as early versions of the final application should be designed appropriately and implemented in Ada. When prototypes are developed, however, often little attention is paid to effectively using software engineering features. Certainly, prototyping should never be an excuse for hacking, and one should always have honorable purposes in mind before starting a prototype. Prototyping typically generates poorly designed and poorly documented code, and there are real dangers in converting this code to be part of the operational software. Generally, code designed for prototyping is unsuitable for supporting an operational mission. Consequently, even when Ada is used for a prototype, the final application may still require appropriate design activities that use software engineering principles.

5.3.2 Simulators and Simulation Languages

Simulators are successfully being developed in Ada. Ada has been extensively used in the flight simulator and flight trainer domain. In the past, the different environments for operational, simulation, and training software resulted in different software developed for each. Ada's portability and modularity of design provide for significant reuse of Ada software in each of these environments, which results in savings of time and money. Ada also is suitable for operations research simulations and modeling, areas traditionally covered by special-purpose simulation languages such as SIMSCRIPT and GPSS. The use of Ada requires some additional effort to provide simulation scheduling and report generation, which are provided by traditional simulation languages. Use of Ada, however, should reduce the life-cycle maintenance costs of these simulation and modeling programs. Another advantage is Ada's tasking (parallel processing). Tasking is effective in simulating logically parallel activities, and it allows multiple processors to be used simultaneously to do the work previously done by expensive, high-powered processors (Law, 1992).

5.3.3 Reuse

Several initiatives are underway in DOD to institutionalize software reuse. This subsection surveys estimates of the economic benefits of software reuse and summarizes some reuse techniques and issues that apply to Ada software development. In addition, Appendix A.1.5 lists several government and DOD reuse repositories.

5.3.3.1 Economic Benefits of Software Reuse

Attempts have been made to quantify the economic benefits of software reuse for software development. Estimates of potential savings range from 20% to more than 90%. Most experts on this topic agree that reuse techniques should be applied across all phases of software development (i.e., requirements definition, design, code production, test, and Post-Deployment Software Support [PDSS] to increase significantly the potential for cost savings. The Software Engineering Institute (SEI) recently published economic models that estimate cost savings resulting from

software reuse under a variety of conditions. The SEI models provide a qualitative analysis of conditions needed to make software reuse economically beneficial.

5.3.3.2 Software Reuse Techniques Applicable to Ada

This subsection samples current techniques in software reuse that can be applied to Ada. These techniques are not mutually exclusive; successful projects have adopted concepts and methods from more than one of the following reuse techniques:

- Development of Generic Components (megaprogramming)
- Design techniques
- Commercial Off-The-Shelf (COTS) software.

5.3.3.2.1 Development of Generic Components (Megaprogramming)

Classification techniques attempt to describe how a software repository is organized so that components may be easily identified and retrieved. Domain analysis, a method by which an application domain is decomposed into component processes, is one such technique. The resulting collection of connected processes may serve as a standard for organizing a reuse library of that domain. Additional information on domain analysis is available in "An Object-Oriented Approach to Domain Analysis" (Shlaer and Miller, 1989). This technique and several similar techniques describe a reuse repository where identifying information is stored in n-tuplets and accessed by a query system. Additional information on this technique can be found in "Classifying Software for Reusability" (Prieto-Diaz and Freeman, 1987).

5.3.3.2.2 Design Techniques

Information hiding is a design technique that allows software costs to be significantly reduced by keeping software changes as localized as possible. This design technique can have a positive impact on software reuse because well-designed Ada packages containing few input parameters (hence, less need for information from the environment external to the component) are more likely to be reusable. A common example would be the abstraction or hiding of device-dependent logic from other portions of the program so that the other portions may be reused easily with different devices.

5.3.3.2.3 Commercial-Off-The-Shelf Software

Reusing COTS software is another technique that can be effectively used in an Ada software development. When COTS software meets system requirements and appropriate licensing rights can be acquired, its use can reduce significantly the costs and schedules associated with system development and acquisition. The availability of Ada bindings, described in Appendix H, provides a framework and means for effectively integrating COTS software into an Ada software development effort. Ada policy encourages the use of COTS software (independent of its underlying language

implementation) as long as the acquiring program office does not modify or maintain the COTS software.

5.3.3.3 Management Issues

Among the incentives for increasing software reuse are improved productivity because less code development is required and improved quality because previously tested code is used. Factors that inhibit software reuse include lack of a standard software architecture; lack of trust in code developed elsewhere; the desire to use the latest innovative languages, tools, and approaches; and lack of knowledge about or difficulty in obtaining information on available tools, software, or repositories.

Domain analysis is a method of identifying the commonalities and variations of existing systems within the same application area to determine reuse candidates. Domain analysis should be performed as part of the requirements analysis to gain a better understanding of existing systems in terms of common functionality and to identify components from existing systems that can be reused in the current development. Domain analysis conducted during requirements analysis uncovers missing requirements. Addressing these missing requirements early helps to stabilize the requirements package.

The issue of data rights is sometimes difficult to address. A piece of software written by an employee of a company on company time generally belongs to the company. In addition, if software development is contracted out, the contract should specify who owns the software upon delivery. Program Managers should ensure, when reusing software from another source, that they have obtained the appropriate ownership or licensing rights. Another data rights issue that should be considered is the responsibility for software that does not work correctly. This is an issue that the Government faces with Government-Furnished Equipment (GFE). When the software is GFE and it does not work, the Government may be responsible for any milestone slips or additional cost resulting from the software problem.

5.3.4 Reengineering

Reengineering refers to the redesign of one or more elements of a software system to improve a system's functionality. Conversion of a software system from an existing programming language to Ada is considered a form of reengineering. A variety of reengineering products are available in the commercial marketplace.

Organizations and Program Managers responsible for long-term maintenance of DOD software systems should understand the relevance and potential benefits of the reengineering concept. From a management perspective, use of automated tools is the key to the reengineering process. Where tools are available, system reengineering can be performed quickly and economically. Reengineering may be

particularly advantageous in situations where large libraries of non-Ada code exist. Because all new systems and any major modifications to existing code must be developed in Ada, organizations with software maintenance responsibilities will be required to maintain expertise in both Ada and the programming languages of their existing systems. In addition, these organizations face potentially complex and costly integration and cohabitation problems as they attempt to develop and operate hybrid systems consisting of Ada and non-Ada code. When automated tools are available, it may be cost-effective to reengineer all existing code to Ada and thereby eliminate the need to maintain long-term programming expertise in other languages. The costs and technical risks of interfacing Ada and non-Ada code also would be eliminated by such a strategy.

Ada also can provide excellent interfaces to many HOLs and assembler, machine language, and system services. Although this capability is vendor dependent, it can allow for an effective transition strategy to Ada. Sections of existing non-Ada code that require little or no change can be easily incorporated into the Ada application.

Reengineering efforts, however, should be undertaken cautiously. Some reengineering techniques neither provide easily readable Ada code nor take advantage of Ada features. No firm guidance can be given as to whether reengineering is the right option for a particular project or organization. As noted, the commercial market is well stocked in reengineering products. Program Managers need to familiarize themselves with the reengineering marketplace to determine whether reengineering represents a viable and cost-effective path for their organizations.

5.3.5 Reverse Engineering

The purpose of reverse engineering is to automatically extract the design information for a system from the existing system source code. Currently, reverse engineering is used primarily to generate documentation products to help manually support and modify systems by using existing source code. The ultimate goal of reverse engineering, however, is to extract design information from the existing system in a standard design format with an automated tool. A functionally equivalent replacement system could then be automatically generated by the tool selected. Under this scenario, any required change to the system would be accomplished at the design-specification level. Several products are emerging on the market to support reverse engineering.

Industry observers generally agree that this type of capability will soon become available. The emergence of this type of tool will provide Program Managers with an additional positive option to deal with the hybrid Ada and non-Ada code maintenance problem. It will soon be possible to improve the basic design and

structure of an existing system, incorporate new requirements, and then regenerate the entire system. As with the reengineering market, there is extensive commercial activity in the area of reverse engineering products. Program Managers need to maintain familiarity with the available technology.

5.3.6 Open Systems Environment

The following discussion is based on the U.S. Department of Commerce, National Institute of Standards and Technology (NIST) Application Portability Profile (APP), the U.S. Government's Open System Environment Profile OSE/1 Version 2.0; and the DOD Technical Architecture Framework for Information Management (TAFIM) Version 1.1

Federal information systems initially developed from isolated islands of computing. Through progressive changes, these individual systems became connected by common users and common information needs. These systems are now well on the way to migrating toward computing environments that consist of distributed, heterogeneous, networked applications, databases, and hardware. The concept of a Federal computing environment that is built on an infrastructure defined by open, consensus-based standards is well on its way to becoming a de facto means of organizing these systems. Such an infrastructure is called an Open Systems Environment (OSE).

An OSE encompasses the functionality needed to provide *interoperability, portability,* and scalability of computerized applications across networks of heterogeneous, multivendor hardware/software/communication platforms. The OSE forms an extensible framework that allows services, interfaces, protocols, and supporting data formats to be defined in terms of nonproprietary specifications that evolve through open (public), consensus-based forums.

5.3.6.1 Benefits of an Open Systems Environment

From the perspective of users and technologists alike, an OSE consists of a computing support infrastructure that facilitates the acquisition of applications with the following attributes:

- Execute on any vendor's platform
- Use any vendor's operating system
- Access any vendor's database
- Communicate and interoperate over any vendor's networks
- Are secure and manageable
- Interact with users through a common human-computer interface.

In more technical terms, an OSE is a computing environment that supports portable, scalable, and interoperable applications through standard services, interfaces, data formats, and protocols. The standards may consist of international, national, industry, or other open (public) specifications. These specifications are available to any user or vendor for use in building systems and products that meet OSE criteria.

Applications in an OSE are scalable among a variety of platform and network configurations, from stand-alone microcomputers, to large distributed systems that may include microcomputers, workstations, minicomputers, mainframes, and supercomputers, or any configuration in between. The existence of greater or fewer computing resources on any platform will be apparent to users only in the context they affect the application's speed of execution (e.g., the time it takes to refresh screens, retrieve data, and/or process data).

Applications interoperate by using standard communications protocols, data interchange formats, and distributed system interfaces to transmit, receive, understand, and use information. The process of moving information from one platform, through a Local Area Network (LAN), Wide Area Network (WAN), or combination of networks to other platforms should be transparent to the application and the user. Location of other platforms, users, databases, and programs should also be transparent to the application.

In short, an OSE supports applications through the use of well-defined components, a plug-compatible technology or building-block approach for developing systems.

Unfortunately, *not enough standards are in place to define an OSE completely.* Standards organizations are working on this problem, but much effort is still needed. As technology changes, some standards will become obsolete and other new ones will be required. Organizations can still accomplish a great deal in moving toward an OSE by selecting specifications that will provide greater openness over time.

5.3.6.2 Open Systems Environment Reference Model

The Institute of Electrical and Electronics Engineers (IEEE) POSIX Working Group P1003.0 describes an OSE Reference Model (RM) that is closely aligned with the APP that provides a framework for describing open systems concepts and defining a lexicon of terms that can be agreed upon by all interested parties. Figure 5-2 illustrates the OSE/RM.

Two types of elements are used in the model: entities consisting of the application software, application platform, and platform external environment, on the one hand, and interfaces including the application programming interface and external environment interface on the other hand.

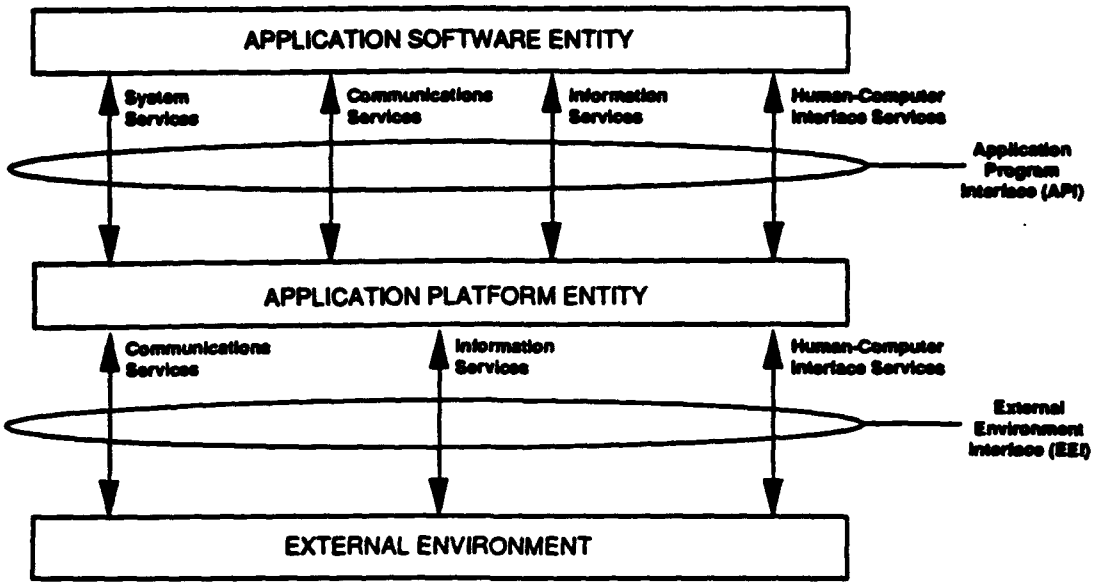


Figure 5-2. Open Systems Environment Reference Model (OSE/RM)

The three classes of OSE/RM entities are described as follows:

- *Application Software* includes data, documentation, and training.
- *Application Platform* consists of a collection of hardware and software components that provides the system services used by application programs.
- *Platform External Environment* consists of those system elements that are external to the application software and the application platform (e.g.,services provided by other platforms or peripheral devices).

Two classes of interfaces in the OSE/RM are described in the following paragraphs.

- *Application Program Interface (API)* is the interface between the application software and the application platform. Its primary function is to support portability of application software. An API is categorized according to the types

of service accessible through that API. The four types of API services in the OSE/RM are:

- Human-Computer Interface Services
- Information Interchange Services
- Communication Services
- Internal System Services.

- *External Environment Interface (EEI)* is the interface that supports information transfer between the application platform and the external environment. Consisting chiefly of protocols and supporting data formats, it supports interoperability to a large extent. An EEI is categorized according to the type of information transfer services provided. The three types of information transfer services are those to and from:

  - Human users
  - External data stores
  - Other application platforms.

5.3.6.3 Application Portability Profile Service Areas

A selected suite of specifications that defines the interfaces, services, protocols, and data formats for a particular class or domain of applications is called a profile. The Application Portability Profile (APP) integrates industry, Federal, national, international, and other specifications into a Federal application profile to provide the functionality necessary to accommodate a broad range of Federal information technology requirements. The APP is directed toward assisting managers, project leaders, and users in making an informed judgment regarding the choice of specifications to meet current requirements.

The services defined in the APP tend to fall into seven broad service areas. These service areas are:

- Operating System Services
- Human-Computer Interface Services
- Software Engineering Services
- Data Management Services
- Data Interchange Services
- Graphics Services
- Network Services.

Figure 5-3 illustrates where each of these seven service areas relates to the OSE Reference Model. The software engineering services are not shown as they are

applicable in all areas. Two supporting services are integrated within and permeate the other seven service areas. In many cases, separate specifications are not available for these supporting services within each of the seven service areas. These two services are:

- Security Services
- Management Services.

The following sections briefly define each service area.

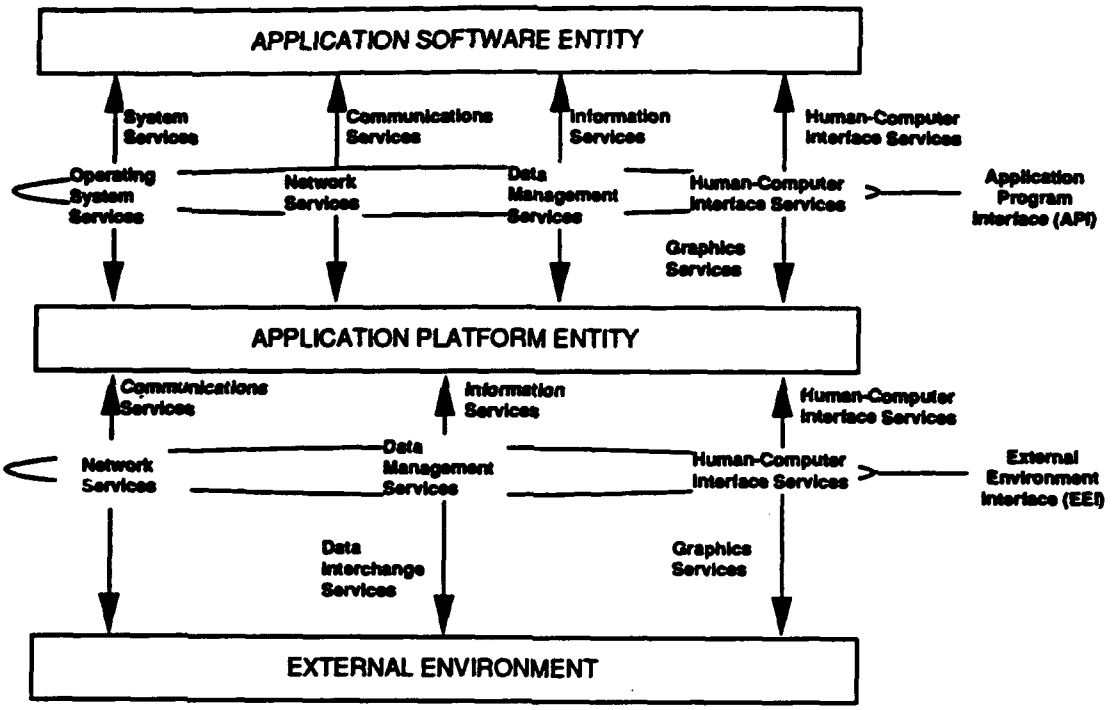


Figure 5-3. APP Service Areas and the OSE/RM

5.3.6.3.1 Operating System Services

Operating system services are the core services needed to operate and administer the application platform and provide and interface between application software and the platform. These core services consist of kernel operations, commands and utilities, realtime extensions, and system management.

5.3.6.3.2 Human-Computer Interface Services

Human-Computer Interface (HCI) services define the methods by which people may interact with an application. Depending on the capabilities required by users and the applications, these interfaces may include client-server operations, object definition and management, window management, dialogue support, and multimedia.

5.3.6.3.3 Software Engineering Services

The objective of open systems is to produce and use portable, scalable, interoperable software. Software engineering services provide the infrastructure to develop and maintain software that exhibits the required characteristics. Standard programming languages and software engineering tools and environments become central to meeting this objective.

5.3.6.3.4 Data Management Services

Central to most systems is the management of data that can be defined independently of the process that creates or uses it, maintained indefinitely, and shared among many processes. Data management services include data dictionary or directory services, DataBase Management System (DBMS) services, and distributed data services.

5.3.6.3.5 Data Interchange Services

Data interchange services provide specialized support for the exchange of information including format and semantics of data entities between applications on the same or different (heterogeneous) platforms. Data interchange services currently include document services, graphics data services, and product data interchange services.

5.3.6.3.6 Graphics Services

Graphics services provide functions required for creating and manipulating displayed images. These services include display element definition and management and image attribute definition. The services are defined in specification for describing multidimensional graphic objects and images in a form that is independent of devices. Graphics security services in this area include access to, and integrity of, functions that support the development of imaging and graphics software and image data.

5.3.6.3.7 Network Services

Network services provide the capabilities and mechanisms to support distributed applications requiring data access and applications interoperability in heterogeneous, networked environments. These services include the data communication, transparent file access, PC or microcomputer support, and remote procedure call.

5.3.6.3.8 Security Services

Security services are provided to support distribution and integrity of information and to protect the computing infrastructure from unauthorized access. These services include operating system security services, HCI security services, programming security services, data management security services, data interchange security services, graphics security services, and network security services.

5.3.6.3.9 Management Services

Management services are integral to the operation of an OSE. They provide the mechanisms to monitor and control the operation of individual applications, databases, systems, platforms, networks, and user interactions with these components. Management services enable users and systems to become more efficient in performing required work. Management is better able to streamline the operation, administration, and maintenance of open systems components. These services include fault management and control services, configuration control services, accounting services, and performance monitoring services.

5.3.7 System Portability

Portability refers to the ease with which software can be transferred from one computer system or environment to another. Over the life of an application program, the host development environment frequently changes because of hardware upgrades, modernization, and the transition of support from a contractor to a government facility. Portability must be considered when selecting or developing tools and software.

Porting from one hardware environment to another presents special portability problems because word sizes may be different, which affects numeric representation and accuracy, and architectures are different, which affects interrupt processing, interfaces, and bus commands. Porting from one software environment to another can also pose special problems. For example, operating system calls and interprocess communications differ significantly among different operating systems.

Ada code is perhaps more portable than any other language because of Ada's validation suite and the capability to isolate system dependencies through Ada packages. Because of a comprehensive conformance test suite involving more than 4,000 tests, a high level of uniformity is achieved among validated Ada compilers. Furthermore, hardware and software dependencies can be isolated in a small number of modules. With appropriate isolation of these dependencies, the number of modules requiring change to support porting from one compiler to another can be very small.

NobelTech recently conducted a portability exercise. In this exercise, about 6,000 Ada modules written for the Motorola 68020 chip were ported to the Reduced Instruction Set Computing (RISC) architecture. Of these 6,000 modules, only about 30 modules required source code changes for the port. Appendix I, Section I.8, highlights the complexities involved in achieving portability in the Tactical Aircraft Mission Planning System (TAMPS).

5.3.8 Ada Compared to Assembler

Coding with High Order Languages (HOLs) such as Ada has important benefits when compared to coding in Assembly. McDonnell Douglas demonstrated these benefits for the computation of the Structural Filter as part of the F-15 integrated flight control system, which was flown in September 1984. Appendix L provides this example.

From this example, it is clear that the Assembler code would be very prone to error. A highly skilled Assembler programmer is likely to make several errors just coding this small example. It is virtually impossible for a nonprogrammer to look at the code and make sense of it. In contrast, the Ada code looks almost identical to the code coming out of the system specification; therefore, any errors are likely to be detected easily during design reviews. Code generated in Ada is more reliable and has a higher quality than code generated in Assembly. Incidentally, McDonnell Douglas projected an order of magnitude improvement in programmer productivity using Ada rather than Fixed Point Assembly Language for design, coding, and testing.

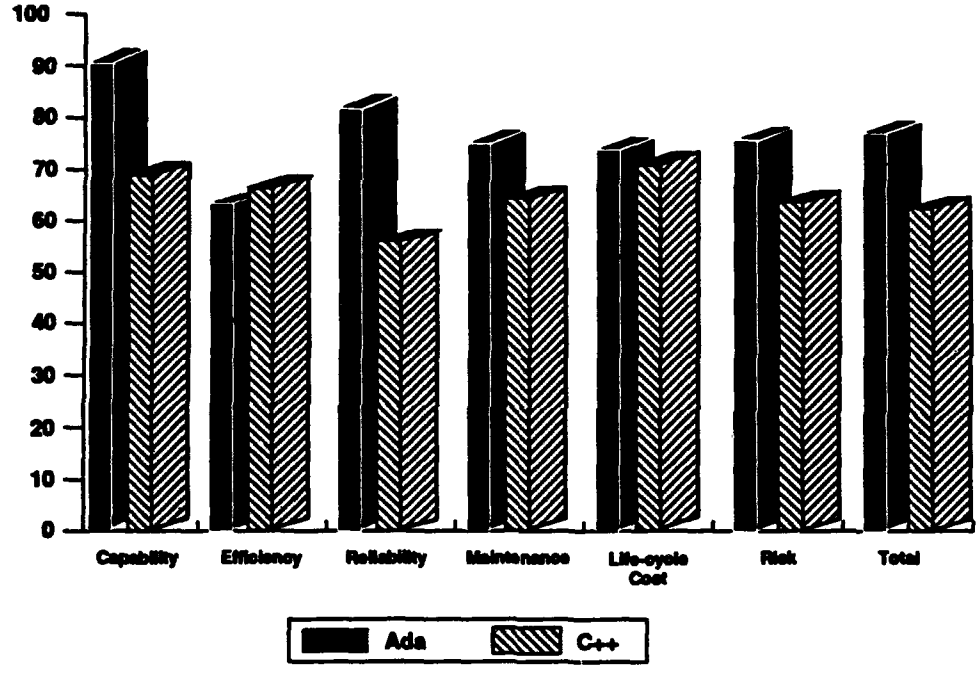
Project Managers expect that Assembly code can be made to execute much faster than Ada code. In most situations, this is not true. Today, optimizing compilers can generate Ada executable code that is much faster than that done in Assembly. A highly skilled Assembly programmer will not normally find all of the places where optimization can manually be coded; an optimizing compiler will. Occasionally, a highly skilled Assembly programmer can generate slightly more efficient code, especially in tight loops where there are interfaces to nonstandard hardware. Ada provides an excellent interface to machine-level code so that this can be done when required.

5.3.9 Ada Compared to C, C++

In 1991, the Air Force conducted a Business Case Analysis to compare Ada to C++ to determine under what circumstances a waiver to the DOD Ada requirement might be warranted for use of C++, particularly in the DOD's Corporate Information Management (CIM) Program. This report examined quantitatively the availability of tools, language selection methodologies, cost, and the emerging impact of fourth-generation technology. Each study reached the same conclusion: there is no compelling reason to waive the Ada requirement to use C++.

The report summary stated that it is impossible to make a credible case for the existence of *more cost-effective* C++ systems compared to Ada. Business cost-effectiveness data collected for the study showed that Ada provides development cost advantages on the order of 35% and maintenance cost advantages on the order of 70%.

A March 1991 SEI study compared Ada to C and C++ in the areas of capability, efficiency, reliability, maintainability, life-cycle cost, and risk (Carnegie-Mellon University, 1991). Ada 83 compared quite well to C++ as Figure 5-4 depicts.



Source: Carnegie-Mellon University/Software Engineering Institute, 1991

Figure 5-4. Comparison of Ada to C++ (SEI, 1991)

A report from New York University in April 1992, titled *Contrasts: Ada 9X and C++* indicated that:

> Ada 9X had similar advantages over C++, particularly when software costs were examined over the lifetime of the software system. Ada 9X was also found to be superior in terms of safety and reliability. A copy of this report can be obtained through the AdaIC, which is described in Appendix A, Section A.1.1.

5.3.10 Mixing Ada With Other Languages

One of the major challenges to the more widespread use of Ada within DOD is the large number of non-Ada systems that are being reengineered or upgraded. The number of new starts, as compared to upgrades, is very low. Even for new starts, often the strategy is to make the greatest possible use of existing code.

Several Ada features support interfaces to other languages. Interfaces to languages such as C, FORTRAN, COBOL, Pascal, and Assembly are common. Such interfaces are vendor dependent and may not be portable from one compilation system to another. If such capabilities are desired, one should check with the compiler vendor for support of Pragma INTERFACE for the desired language(s).

The National Aeronautics and Space Administration (NASA) has been successful in mixing Ada with other languages. They have a considerable investment in legacy software written in FORTRAN, C, and Pascal. By making use of an Ada infrastructure, NASA is able to reuse much of this legacy software and take advantage of the software engineering benefits of an Ada architecture.

5.4 PARADIGM SHIFTS FOR EFFECTIVE SOFTWARE ENGINEERING

Correct orientation in education and training is critical in tapping the strengths of Ada that can help us achieve a paradigm shift to effective software engineering.

In its shallowest sense, Ada *is* just another programming language. If software developers are introduced to it in this context, it will be of little benefit. Taught in the context of a tool to support good software engineering practices, however, it can be of great benefit. In short, it is the paradigm shift, the cultural change, in how to develop software systems that is imperative and will bring the most gains. Ada is simply one of the most effective tools at our disposal to support this shift. To support this paradigm shift, education and training programs must focus on software engineering and teach Ada in the context of a tool to support it.

Section 6

Lessons Learned

This section summarizes the collective set of lessons learned on several Department of Defense (DOD) and commercial Ada projects, including the following:

- Stratcom-Computer Center, Offutt Air Force Base
- Wells Fargo Nikko Investment Advisors
- B-2 Aircrew Training Devices
- Boeing Military Aircraft (Wichita, Kansas)
- Coulter Electronics: Ada for Cytometry
- AN/UYS-2A Project
- Ada Experience at the Naval Research and Development Center
- Tactical Aircraft Mission Planning System
- Advanced Field Artillery Tactical Data System
- AN/BSY-2 Submarine Control System
- Ada Language System/Navy Full-Scale Development Program
- Avionics Project for an Airborne Command, Control, and Intelligence Application
- PEO-SSAS, PMS-414, SEA LANCE
- Navy World Wide Military Command and Control System (WWMCCS) Site-Unique Software Project
- Event-Driven Language/COBOL-to-Ada Conversion Program
- Shipboard Gridlock System with Auto-Correlation
- Combat Control System MK2

- P-3C Update IV Ada Development
- Standard Financial System Redesign
- Reconfigurable Mission Computer Project
- Intelligent Missile Project.

Appendix I provides a full description of these projects and the lessons learned on each. This appendix also provides a matrix showing the lessons learned by specific category and project. As review of this appendix shows, most of the problems encountered were management-related, not Ada-specific, problems. In addition, some problems recur across all of these projects, such as:

- Lack of training and/or experience
- Failure to take a risk engineering approach
- Improperly specified contract requirements for software-related items and processes
- Inadequate estimates of resources and/or facilities needed
- Immaturity of Ada development tools and environments
- Insufficient incremental testing or lack thereof.

The subsections below highlight a few of the lessons common to several of the projects in the areas of standards and policy, project management, development process, corporate knowledge and software development experience, training, resources and facilities, support environment tools, reuse, and project costs.

Before undertaking any software-intensive system development, the reader should review the matrix in Appendix I, and the Project Manager should study the detailed project descriptions in this appendix so as to benefit from the lessons learned.

6.1 STANDARDS AND POLICY

Lesson Area Summary: Review of the project lessons shows the necessity for establishing a policy to ensure that planning and monitoring of software development occur early in the development process. The lessons also highlight the importance of incorporating the critical elements of the Military Standards into the acquisition package (e.g., the Request for Proposals). In addition, the lessons suggest that

policies be established to require incremental development, use of metrics from the beginning of the project, and development of a common style guide for use across development teams.

Continuing Challenges: DOD acquisition policies have recently been revised. It is still too early to assess the overall impact of these revisions on acquisition management practices. Emerging policies and standards that deal with software reuse, Open Systems Architecture (OSA), information system security, tools, and use of Nondevelopmental Item/Commercial-Off-The-Shelf software will directly affect software acquisition practices.

6.2 PROJECT MANAGEMENT

Lesson Area Summary: Nearly every lesson learned in these projects related to project management, as the matrix in Figure I-1 in Appendix I shows. Among the recurring lessons in this area is the need for up-front planning and close monitoring of the project. Also evident, and related to up-front planning, is the necessity for ensuring that adequate facilities and resources are available.

Continuing Challenges: Within the area of project management, new and strong emphasis is being placed on the early assessment and evaluation of true life-cycle costs. Among the topics being addressed are evolutionary (lower risk) developments; risk planning and control; complete metrics programs; increased use of commercial, nonproprietary software, hardware, and networking interface standards; optimized software portability; and design for reuse. As in the past, the need to ensure adequate staffing, resources, and facilities to accomplish the work must be anticipated and addressed. These key ingredients all need to be incorporated into a well-thought-out early planning effort. Once execution begins (based on a complete Work Breakdown Structure [WBS]), status and progress towards interim deadlines must be continuously monitored.

6.3 DEVELOPMENT PROCESS

Lesson Area Summary: The most significant lesson learned across projects was the importance of applying sound systems engineering and software development practices and principles at every stage of the project. Of particular importance is strict adherence to configuration management and Quality Assurance practices. On several projects, use of a consistent methodology and adoption of a risk engineering approach were mentioned as key to a successful development process.

Continuing Challenges: Greater flexibility in the development process is being allowed in today's acquisition environment. The days of starting software developments from scratch and blindly imposing a waterfall development process have passed. Advances in object-oriented design, OSA interfacing, and legacy

software availability enable systems to evolve through reuse and modular upgrades. The willingness to actively pursue "joint" or "shared" developments should lead to lower risk, decreased costs, and improved quality in meeting mission requirements.

6.4 CORPORATE KNOWLEDGE AND SOFTWARE DEVELOPMENT EXPERIENCE

Lesson Area Summary: In this area, the lessons learned emphasize that both Government and contract developers must understand the project requirements and adhere to them. Corporate knowledge and software development experience also are needed to establish schedules, determine at what point full-blown coding should begin, and identify the resources available to meet system requirements.

Continuing Challenges: Knowledge of the corporate domain and ancillary software development experience continue to be critical to the generation and deployment of software-based systems. Such experiential data and information need to be captured as they are generated, saved in the appropriate context, and recalled when needed. Armed with this collective knowledge, planners can make informed estimates of the what, how, when, and why for new undertakings. The goal here is to transition both the engineering and management of systems development from the worst case "initial" to the "optimizing" or continuous process improvement level. Such a conversion will ensure best-quality products and maximum potential to anticipate future change.

6.5 TRAINING

Lesson Area Summary: On several of the projects, the need for Ada-specific training was noted because most of the experienced personnel have little or no experience with Ada and modern software engineering practices. This need for training applies to both technical and management personnel. Project experience suggests that hands-on training should be conducted as close as possible to development or during development.

Continuing Challenges: Ada-specific training problems should decrease over time. Training in application software engineering process methods and use of Computer-Aided Systems Engineering (CASE) technology tools, however, needs to be addressed for both technical and management personnel.

6.6 RESOURCES AND FACILITIES

Lesson Area Summary: As mentioned above, review of the project lessons indicates that the initial estimates of the resources and facilities needed often were inadequate. Project experiences show the necessity for having developers identify the tools to be used in both system analysis and development to ensure adequate resources will be available to meet requirements.

Continuing Challenges: Resource and facility planning must address both development and post-deployment maintenance needs. For larger, geographically distributed developments, the lowest risk solution is to standardize as many of the support resources as possible, including tools, exchange media, documentation, processes, configuration management, and environments. Managers need to develop resource utilization estimates before beginning projects, and they need to monitor utilization closely to track actual consumption against projections.

6.7 TOOLS

Lesson Area Summary: For large, geographically dispersed projects, the lessons learned show that common support tools should be required. Use of common tools allows problems to be identified quickly, workarounds made only once, and results entered into a shared electronic reporting system. In addition, before committing to large projects, the methods and tools should be exercised; the team must be well trained in the use of the supplied tools; and the tools must work as advertised. Project experience also indicates that use of automated tools should be mandatory for large software undertakings and that development tools are essential.

Continuing Challenges: The number of vendor tool products continues to increase as the industry matures and enters its second decade. Many products on the market, however, fail to meet the goal of total life-cycle support. Integrated data and information exchange, along with interoperability among differing vendor tools, remains elusive. For CASE tools, project experience has also shown that unless proper evaluation, selection, training, and management commitment occurs, the products are sometimes abandoned because of poor performance and lack of utility.

6.8 REUSE

Lesson Area Summary: The project lessons show that Ada facilitates reuse and that planning for and designing in reuse yield long-term benefits. It was noted that development and maintenance time can be reduced significantly by capitalizing on reuse. However, project experience suggests that large-scale software component reuse will depend on achieving more technological progress.

Continuing Challenges: Within the past 2 years, concern about the affordability of systems and the need for evolutionary upgrades has escalated. In response to that concern, the efforts to promote reuse have increased dramatically. Tool technology has begun to provide automated ways of designing for reuse and evaluating legacy software assets. As with CASE technology, key ingredients for success include corporate commitment, planning, resource investment, engineering and management discipline, and correct tool selection and usage.

6.9 PROJECT COSTS

Lesson Area Summary: On several projects, the effect of inadequate initial estimates of the needed resources had cost implications. In some cases, additional funding was needed for support hardware and facilities and for training as well as to accommodate schedule delays.

Continuing Challenges: The impending cuts in the overall DOD budget will make it necessary to learn how to accomplish more with less. The number of start-from-scratch projects will decrease whereas the need to plan carefully for smaller upgrades of existing systems will increase. There will be a much greater emphasis on demonstrating a new concept before allowing full production to proceed. In such an environment, funds will need to be invested to ensure developers fully understand existing products and their capabilities. From a software perspective, investment in domain-specific software reuse needs to be accelerated. In addition, the downscaling of worldwide threats should cause reallocation of funds away from only mission-driven requirements to a more even-handed set of requirements that addresses mission, quality, and affordability.

Section 7

Future Directions

Evolving technology is valuable to Program Managers as the latest initiatives become the state of the practice. Several important initiatives are examining ways to improve the software development process, increase productivity, increase quality, and reduce costs. These initiatives will be important to the Department of the Navy (DON) in the near future.

This section provides a description of the following initiatives, which warrant attention:

- Ada 9X
- Ada Reuse
- Corporate Information Management (CIM)
- Integrated Computer-Aided Software Engineering (I-CASE) tools
- Next Generation Computer Resources (NGCR)
- North American Portable Common Tool Environment (PCTE) Initiative (NAPI)
- Portable Common Interface Set (PCIS)
- Software Engineering Institute (SEI)
- Software Executive Official Council (SEOC)
- Software Technology for Adaptable, Reliable Systems (STARS)
- Tactical Advanced Computer-4 (TAC-4) and TAC-5 Procurement
- Technology plans, including:
  - Software Action Plan (SWAP)
  - Software Technology Strategy Document
  - DON Reuse Implementation Plan and Guide
  - DON Information Management Strategic Plan
  - DON Software Process Improvement (SPI) Plan
- DON Technology Pilots, including:
  - I-CASE Pilots
  - Functional Process Improvement (FPI)
  - SEI Pilots
  - STARS Demonstration Pilot.

7.1 Ada 9X

The tenets of both the American National Standards Institute (ANSI) and the International Organization for Standardization (ISO) require that each standard be periodically revisited. Ada 9X is the effort to perform this function for the Ada programming language.

7.1.1 Background

The Ada language standard, ANSI/Military Standard (MIL-STD)-1815A, was published in 1983. Starting in 1984, the number of available Ada development facilities began to increase. As Ada was rigorously used in several projects, a series of omissions, limitations, and minor errors were identified. In January 1988, the Ada Joint Program Office (AJPO) asked the Ada Board for a recommendation on how to resolve this situation.

In September 1988, the Ada Board delivered its report, which recommended revising the language. To accomplish this task, the Ada 9X Project Office was established under the direction of Christine M. Anderson at Eglin Air Force Base (AFB), Florida, and was relocated to Kirtland AFB, New Mexico, in July 1992. The goal of the project is to revise Ada 83 and effect a smooth transition from Ada 83 to Ada 9X (ANSI/MIL-STD-1815B). During the project, a public survey was conducted to solicit revision requests, and more than 750 revision requests were received. Several international workshops were convened to review and rank those inputs.

Changes have been constrained by the overall objective of minimizing the negative impact and maximizing the positive impact on the Ada community. Upward compatibility between Ada 83 and Ada 9X is a high-priority goal. The effect on managers, programmers, vendors, educators, authors, and various application domains will be considered throughout the revision process.

The revision will include only those changes that improve the usability of the language while minimizing the disruptive effects of changing the standard. The revision process will continue and will include various forms of public scrutiny such as conferences, electronic mail comments, and draft documentation. The draft Ada 9X standard will be released for voting by ANSI and ISO in September 1993. ANSI and ISO approval of the revisions is expected in 1994.

7.1.1.1 Requirements

The proposed revision requirements, which were completed in December 1990, are grouped into the following categories:

- *General requirements*. Collection of small defects in the language with the structure and format of the standard retained
- *Real-time requirements*. Precise control over when an action occurs
- *Systems programming requirements*. Machine operations, data interoperability, interrupt entry binding, and operations on pointers

- *Safety-critical trusted requirements.* Ability to analyze generated code for certification and to provide correspondence between the source and the generated code

- *Support of programming paradigms.* Subprogram manipulation, data storage control, recompilation, object-oriented programming support, and generic modifications

- *Parallel or distributed processing (capability currently does not exist).* Distribution of single programs, distribution of an Ada system, remote communications, and configuration control

- *Information systems.* Currency quantity handling, character set compatibility, interface to Database Management Systems (DBMSs), and common data structures

- *Scientific and mathematical applications.* Location of point and data storage

- *International user requirements.* Topics such as international character sets.

7.1.1.2 Revision Activities

The requirements have been mapped (1991–1992) into language solutions, and the wording in the standard will be revised by September 1993. Three major enhancements include support for object-oriented programming, programming-in-the-large, and lightweight synchronization.

7.1.2 Ada 9X Transition Activities

Transition activities involve management, programmers, vendors, and Ada Compiler Validation Capability (ACVC) test suite revision and policy.

7.1.2.1 Managers

To help managers transition to Ada 9X, two Ada 9X workshops for managers will be conducted before the formal standard approval—one for mid-level management and one for executive-level management. Transition issues and strategies will be discussed.

A short (approximately 15- to 20-minute) videotape that discusses the language in terms of corporate benefits and policy issues will be developed for managers and a concise guide to practical steps for transitioning to Ada 9X from both non-Ada and Ada 83-oriented organizations will be developed. The guide will be similar to the Ada Adoption Handbook for Ada 83 developed by SEI. It will include a discussion

of the benefits of using Ada 9X from a manager's perspective, tips on tool selection, and a summary of policy.

7.1.2.2 Programmers

To help Ada 83 programmers transition to Ada 9X, an *Ada 9X Programmer's Guide* will be developed. This guide will highlight, chapter by chapter, the changes between Ada 9X and Ada 83 and discuss programming strategies that use new features. Any incompatibilities between Ada 9X and Ada 83 will also be noted, and straightforward modifications to Ada 83 code will be provided to transition to equivalent Ada 9X code. Suggested Ada 83 coding practices to facilitate the transition to Ada 9X also will be discussed for those programmers who are continuing to use Ada 83 on existing projects. A 1-hour videotape also will be developed that will highlight the changes to the language and will feature opportunities for use as well as programming examples.

7.1.2.3 Vendors

During the Ada 9X revision process, several workshops for vendors will be conducted. The purpose of these workshops will be to allow vendors to closely track the revision and to provide feedback on implementability to Ada 9X teams. An electronic vendor bulletin board has been established to allow vendors to interact directly with Ada 9X Project team members. Open and direct dialogue is essential for a timely and effective transition.

7.1.2.4 ACVC Test Suite Revision

AJPO has frozen the Ada 83 test suite (ACVC 1.11). For information only, a baseline for the Ada 9X ACVC Test Suite, 9XBasic, will be released approximately 12 months before ANSI approval. It will eliminate incompatibilities between Ada 83 and Ada 9X and focus on usage-oriented tasks rather than remote fringes of the language.

The first Ada 9X ACVC release will be designated ACVC 2.0 and will cover part of the new Ada 9X features. The Ada 9X test suite will focus on usage. Table 7-1 provides the planned release schedule.

Suite	Objectives Available	Tests Available	Start	End	Certificate Expiration
2.0	2 MAC	3 MA9X	3 MA9X	27 MA9X	36 MA9X
2.1	3 MA9X	9 MA9X	27 MA9X	63 MA9X	75 MA9X

MAC = Months after release of Ada 9X ANSI canvass for voting
MA9X = Months after ANSI approval of Ada 9X

Table 7-1. ACVC Planned Release Schedule

7.2 Ada REUSE

Several efforts (e.g., STARS, CIM, SWAP) are under way to address software reuse. The Department of Defense (DOD) and other agencies (e.g., the Joint Integrated Avionics Working Group [JIAWG], National Aeronautics and Space Administration [NASA]) recognize that software reuse has the potential to yield substantial improvements in the quality and reliability of DOD software systems at a reduced cost. The main objective of all of these efforts is to create an environment in which Program Managers can reuse already developed software components rather than develop new code. The reuse concept, however, raises several issues that must be addressed and resolved, including the following:

- Policy and regulations that inhibit software reuse
- Incentives to developers, Program Managers, and contractors to reuse existing software
- DOD infrastructure to facilitate widespread software reuse
- Cultural change in the areas of software development, acquisition, and support to accept and promote reuse
- Legal and contractual issues
- Work force education in areas of reuse technology
- Technology to support confident composition of software components
- Limited tool support.

7.3 CORPORATE INFORMATION MANAGEMENT

CIM is the initiative through which DOD will integrate and strengthen central management of the Defense Information Management Program. The goal of the CIM initiative is to improve the effectiveness and efficiency of business processes in DOD by integrating and streamlining functional requirements and by using information technology to implement the improved business operations that result.

The Secretary of Defense assigned to the Office of the Assistant Secretary of Defense (OASD[C3I]) the responsibility for establishing an organization to implement CIM throughout DOD. Pursuant to this direction and in accordance with the "Plan for Implementation of Corporate Information Management in DOD," approved by the Deputy Secretary on 14 January 1991, OASD(C3I) established a Directorate of Defense Information (DDI), which is responsible for the following:

- Developing and promulgating information management policies
- Implementing information management processes, programs, and standards
- Integrating the principles of information management into all of DOD's functional activities.

This responsibility applies to information technologies and architectures, software, systems development methods and tools, information technology and data standards, and Automatic Data Processing (ADP) equipment acquisition processes. It does not include equipment and software that are an integral part of a weapon or weapons system and related test equipment.

Application of CIM principles will enable managers of functional activities to streamline business methods and business processes, develop sound business cases and functional economic analyses of their activities and supporting information technology, and provide other improvements in the effectiveness and efficiency of the functional activities. The OASD(C3I) DDI will develop and promulgate guidance on the common models, tools, and methodologies to be used by functional personnel in performing their responsibilities for the management of information related to their functions. CIM also supports the goals of the July 1989 Defense Management Report to the President.

7.4 INTEGRATED COMPUTER-AIDED SOFTWARE ENGINEERING TOOLS

"The Department of Defense (DOD) is committed to establishing a single, common Software Engineering Environment (SEE) for the development of Automated Information Systems (AISs)." This dramatic statement introduces a far-reaching policy memo, the DOD I-CASE Use Policy, signed by the Director of Defense

Information on 27 February 1992. With the establishment of the I-CASE policy, which will be enabled by the complementary I-CASE acquisition program, DOD has made a major strategic commitment to apply Computer-Aided Software Engineering (CASE) technology throughout the largest software development organization in the world, the U.S. DOD.

The I-CASE policy will ultimately result in the modernization and standardization of DOD's numerous software development activities. The I-CASE Use Policy memo states, "It is DOD policy that I-CASE will be used by each Military Department and Defense Agency for all in-house, Government-developed AISs." For contractor-developed systems, the policy requires that AIS contracts include provisions for delivering computer products in a form that can be input into an I-CASE SEE. Preliminary results of a recent survey of selected DOD software development activities indicate that DOD owns at least one copy of virtually every commercial CASE tool in the marketplace. Standardizing on I-CASE will reduce the proliferation of incompatible products the Department must support.

To implement the I-CASE policy, the U.S. Air Force has been designated as the Executive Agent for I-CASE and charged with identifying DOD CASE requirements and establishing an acquisition program to meet those requirements. The Air Force has created the I-CASE System Programming Office (SPO) at Maxwell AFB, Gunter Annex, Alabama, to manage the acquisition. The mission of the I-CASE SPO is to bring I-CASE to fruition as soon as possible so that users throughout DOD will have a standard SEE and associated hardware, training, and technical support services. The I-CASE contract, which is expected to be awarded in late 1993, will be available throughout DOD and to other Federal agencies.

The I-CASE acquisition will provide DOD software development and maintenance organizations with the latest SEE. I-CASE will consist primarily of Commercial-Off-The-Shelf (COTS) hardware and software development components and the necessary run-time licenses to execute the systems developed in the environment. The use of COTS tools in the I-CASE environment will be strongly emphasized. I-CASE will include a central repository for storing all information relating to a specific software project and will support a full range of program development tools that covers each phase of the software life cycle. Additionally, I-CASE will support common management functions extending across multiple life-cycle phases such as program management, Quality Assurance (QA), and configuration management.

Figure 7-1 illustrates the I-CASE technical environment.

DOD is moving rapidly to establish a standards-based computing environment and has identified several standards that will be mandatory. I-CASE must operate under

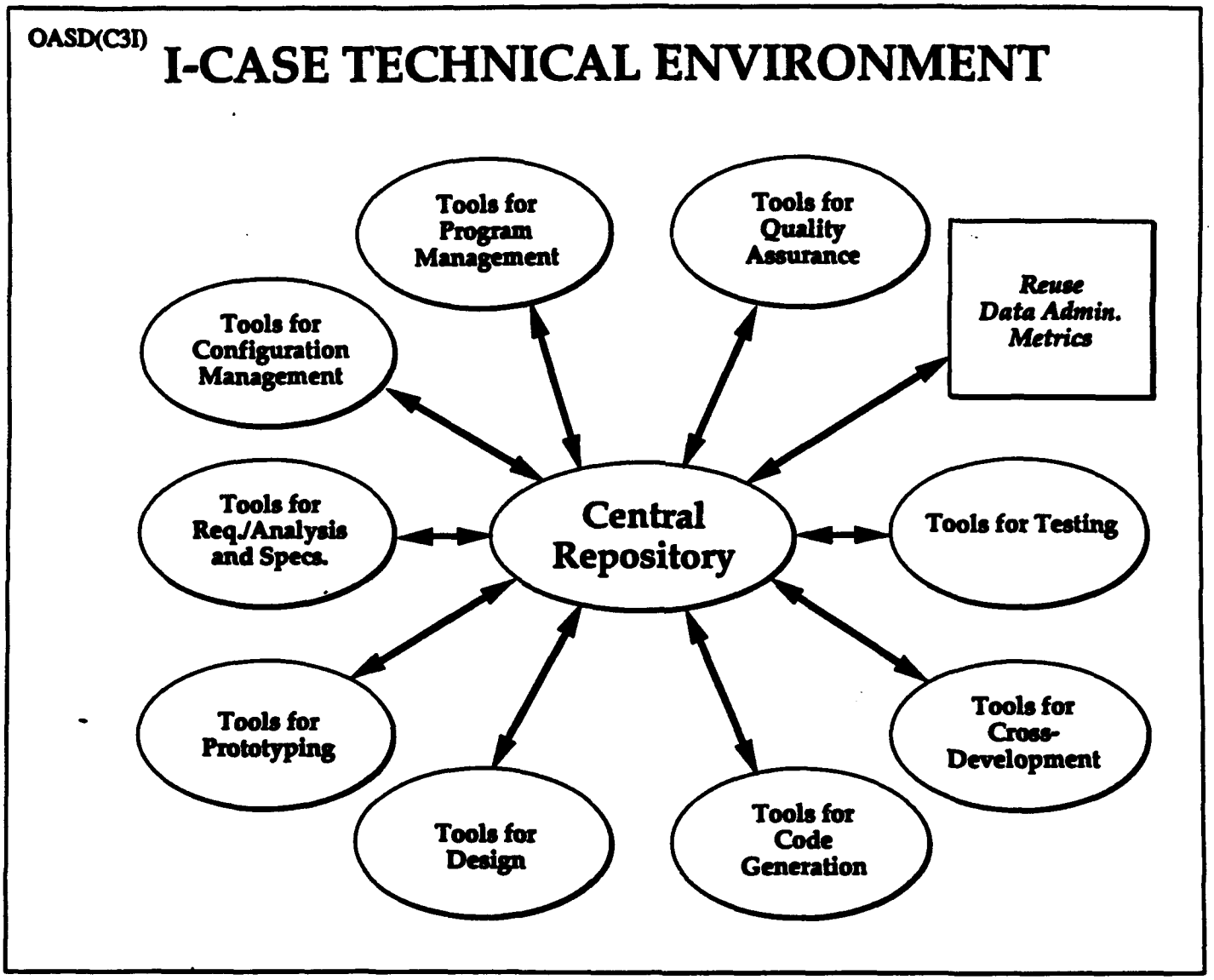


Figure 7-1. I-CASE Technical Environment

these standards and will, in turn, be used to develop applications that execute on open systems platforms. The I-CASE environment will support or meet the following standards:

- *Portable Common Tool Environment (PCTE)* to provide a standard repository interface through the PCTE specification
- *Ada* to generate code in Ada, the DOD standard High Order Language (HOL)
- *Structured Query Language (SQL)* to support an SQL interface to at least two Relational Database Management Systems (RDMSs)
- *XWindows* to support XWindows for end-user applications
- *Portable Operating System Interface for Computer Systems (POSIX)* to support applications executing in a POSIX operating environment
- *Government Open Systems Interconnection Profile (GOSIP)* to provide support for GOSIP.

The I-CASE environment will also comply with the National Institute of Standards and Technology (NIST)/European Computer Manufacturing Association (ECMA) *Reference Model for Frameworks of Software Engineering Environments.*

Public Law 102-396, Section 9070, requires that DOD use the Ada programming language for all applications except where is it cost-effective not to do so. Consequently, Ada has been specified as the only third-generation language that will be supported by the I-CASE environment. Initially, I-CASE will support Ada 83, but when the Ada 9X requirements are approved, I-CASE will evolve to the new version. At a minimum, the I-CASE environment will generate Ada program unit skeletons and the user will be required to complete the functionality. By the end of the 7-year contract, however, we expect I-CASE to generate 100% of the code required for an application.

The I-CASE environment will strongly emphasize the integration of all supported software development tools across each possible dimension of integration (i.e., control, presentation, data, and process). I-CASE will support at least one, but preferably more, of the most widely accepted software development methodologies. DOD systems today are developed by using a wide variety of methodologies ranging from functional and data-driven approaches to state transition and object-oriented methodologies. Support is needed for as many of these as possible; however, the emphasis is on the requirement for an object-oriented methodology. Process and

work flow control of the tools in the environment will be an important feature of I-CASE, and groupware support is expected to be part of any development process.

In addition to providing the traditional functions offered by a SEE, I-CASE will strongly emphasize reuse of previously developed software components. Within I-CASE, the reusability of all software objects in the repository will be supported, from requirements through design to code construction and eventually to testing. Each I-CASE environment will be connected to the DOD Software Repository System (DSRS), a multidomain software reuse library. Each I-CASE environment will also be connected to the DOD Data Repository System (DDRS), which will provide access to common data elements, data definitions, and data models used across DOD. Software metrics, an integral part of the SEE, will capture DOD-specific metrics at each phase of the software life cycle.

The execution of the I-CASE contract will break new ground for a DOD acquisition. DOD recognizes the state of technology of SEEs is evolving rapidly and the I-CASE contract must be innovative in its approach to technology enhancements and upgrades. To ensure that DOD always has current SEE technology, broad requirements have been set for the initial I-CASE delivery. Each requirement in the Request for Proposals (RFP) has been demonstrated in one or more implementations; however, the total of all requirements effectively advances the state of the art. Consequently, I-CASE is structured into tiered requirements, with the mandatory requirements at contract award representing only a subset of the long-term need. At award time, the I-CASE contractor will deliver all mandatory requirements, a subset (large, it is hoped) of the I-CASE "tiered" requirements, and a "migration plan" for incorporating requirements that were not part of the initial delivery. A substantial award fee should encourage adherence to the migration plan.

The contract will require commercialization of the I-CASE environment. Therefore, in addition to providing the I-CASE SEE to DOD and other Government agencies, the winning contractor must also sell the environment commercially. Moreover, it is a DOD goal to make the information repository data model and interface specification open and public information. DOD anticipates that many third-party suppliers of CASE tools will adapt and integrate their products to the I-CASE SEE. DOD has long been the recipient of Government-only technology that frequently approaches obsolescence on the day it is delivered. The I-CASE contract intends to "ride the wave" of commercial SEE development and thus ensure that DOD software developers are provided with the latest technology to support the DOD "customer"—the soldier, sailor, and aviator of the U.S. armed forces.

7.5 NEXT GENERATION COMPUTER RESOURCES

The Next Generation Computer Resources (NGCR) Program is providing Navy air, surface, and subsurface tactical systems Program Managers and system developers with computer hardware and software interface standards that will allow the Navy to transition to commercially based open systems designs. Open systems designs will reduce Navy dependence on the original system suppliers by allowing competition for procurement and modification/upgrades to Navy mission critical systems.

To achieve program objectives, the NGCR program has established working groups in critical computer standardization areas including backplane busses, networks, operating systems, database management systems, graphics, and project support environments. These working groups are composed of NGCR-funded Navy personnel with voluntary participation from industry. The task of each group is to work directly with national and international standards organizations to infuse Navy requirements into the commercial standards. When successful, this allows the Navy to leverage commercial investments by using the resulting open commercial standards in its weapons systems. This modular approach to systems design will allow technology to evolve in a competitive environment and without revolutionary changes in the systems architecture.

The NGCR effort is fostering an Open Systems Architecture (OSA) approach to computer resource acquisition. For OSAs, the internal and external hardware and software interfaces, services, and protocols are well specified; they have undergone public review and have been published and widely accepted as standards by organizations such as the Institute of Electrical and Electronics Engineers (IEEE), ANSI, and ISO; and they are implemented in vendor products. This approach will allow the Navy to take advantage of existing and future industrial competition and innovation on a continuing basis, which will dramatically improve the technical and operational performance of DON computer-based systems.

The NGCR Program is pursuing standards in the following areas (note that the dates provided indicate the planned availability of the completed standards):

- Local Area Network (LAN)—Survivable Adaptable Fiber-optic Embedded Network (SAFENET)*, October 1992
- Backplane (BP)*—FUTUREBUS+, June 1993
- Operating System Interface—POSIX, December 1995
- High Performance Network, September 1996

- High-speed Data Transfer Network, September 1996
- DBMS Interface, September 1998
- Graphics Language Interface, September 1998
- High Performance Backplane (HPBP), TBD
- PSE, suspended.

Of the nine areas, only the BP and the LANs (i.e., items marked with an asterisk) will be actually prototyped and conformance tested as part of the NGCR program. A formal conformance testing capability will be established within DON and will be available for acquisition managers by 1995. The program is encouraging industry to undertake initiatives in this area so as to transition all testing away from the Navy.

7.5.1 Project Support Environment Standards Working Group

Consistent with the objectives of the overall NGCR program, the goal of the Project Support Environment Standards Working Group (PSESWG) is to establish DON standards for PSE interfaces that will enhance the DON's ability to acquire PSEs quickly and cost-effectively. Also consistent with the NGCR program guidance, the PSESWG is a joint team composed of members from DON, other Government organizations, industry, and academia.

PSE interfaces selected for standardization will include data interchange formats and interfaces to the user, a DBMS, life-cycle process management, and a network. Because the objective is to standardize interfaces based on industry standards, the PSESWG work will not select particular tools or products. The group is pursuing the adoption of interfaces with Ada language bindings as well as those for other languages, such as C.

The PSESWG coordinates with several other important groups in the environments community, including SEI, STARS, and NIST, in addition to the other military Services. This coordination is expected to yield the maximum benefit and reduce duplication of effort. Because of the wide range of interfaces to be considered for standardization by PSESWG, the standards are expected to emerge incrementally, perhaps as early as 1994. Although the original plans called for continuing the work until approximately 1998, the NGCR funding situation will not allow the work to continue beyond FY93. The products of the PSESWG to date, most notably the Next Generation Computer Resources (NGCR) Reference Model for Project Support Environments, Version 2.0, 2 September 1993, will continue to be available as the basis for launching future work.

7.5.2 Operating Systems Standards Working Group

The Navy's NGCR program and NASA have selected the POSIX IEEE 1003 standard as the nonproprietary operating system interface standard. The NGCR Operating Systems Standards Working Group (OSSWG) is participating in the IEEE POSIX group to influence the standards so that they will support Navy requirements. POSIX areas of standardization of particular interest to the NGCR program include real-time extensions, Ada bindings, security extensions, and distributed computing standardization features.

7.6 NORTH AMERICAN PORTABLE COMMON TOOL ENVIRONMENT INITIATIVE

The North American Portable Common Tool Environment Initiative (NAPI), a joint technical initiative among Government, industry, and academia, was created to provide a forum to represent North American interests in PCTE and to foster the establishment of a market for the growth of PCTE tools and environments.

7.6.1 Background

During the past decade, the software development community has become increasingly aware of the need for integrated tools that share data and systems that allow tools to interoperate. The full power of CASE tools can only be realized when these tools are integrated into a common, distributed SEE. Such an environment will consist of a framework of common operations that provide basic integration facilities based upon an open common repository with additional support for communication, user interface, and process support functionality.

An integrated environment will provide a platform to facilitate addition of new tools to improve user productivity and software quality. It will benefit both tool developers and users. From the tool developer's perspective, a single set of integration standards will enable development of lower-cost and higher-quality products across multiple hardware platforms. From the user's point of view, the existence of standards will mean that the same type of products will be available from several sources, which will give the user a broader selection and, because of vendor competition, possibly reduce acquisition and maintenance costs. Such interfaces should support the needs of both the defense and non-defense communities.

Among the candidates suggested for such a data interface have been the Common Ada Programming Support Environment (Ada PSE) Interface Set (CAIS-A), developed under the sponsorship of the AJPO (MIL-STD-1838A); A Tool Integration Standard (ATIS), currently under consideration by ANSI working group X3H6; and the PCTE, developed under the aegis of ECMA (ECMA-149). Each of these provides a set of basic services for other software tools, and each of these sets of services (in different ways) supplies some of the capabilities of a "framework."

7.6.2 Focus on PCTE

NAPI will focus on PCTE as a baseline standard because PCTE is now the leading internationally recognized interface standard that appears to meet a reasonable set of framework repository requirements. PCTE does not currently provide all of the functionality that the software development community needs; however, it can evolve toward the needed functionality and NAPI can influence PCTE's evolution in an open process. ECMA and its Technical Committee 33 (TC33) own the standard; however, ECMA is planning to submit it to the Joint Technical Committee 1 (JTC1) in mid-1993 for adoption as an ISO standard. ECMA's ownership, and eventually ISO's ownership, of the standard means that this is a publicly available, nonproprietary standard.

Although no commercial implementations of the full standard exist, Emeraude, Verilog, and Heuristix Systems have implemented a forerunner of the standard (PCTE 1.5). These companies, as well as IBM, Digital Equipment Corporation, ISSI, and EDS Scion, have announced they are implementing the full standard, but no company has provided a firm product release date.

Because PCTE is a technology of considerable importance to both government and industry in North America, NIST, DOD, and the Object Management Group (OMG) have jointly proposed the creation of NAPI with U.S. industry taking a leading role. NAPI will provide a forum for North American interest in PCTE, and give North America a voice to express opinions to TC33, ECMA, and ISO on the future development, evolution, and adoption of the standard and its revisions. NAPI's ultimate goal is a good interface for ISEE technology; the baseline NAPI has chosen to begin working toward this goal is PCTE.

Industry and government in North America will collaborate in NAPI to formalize North American interest in PCTE and to work toward the achievement of a widely used common software tool interface. NAPI will work with other groups that support PCTE, specifically the North American PCTE Users' Group (NAPUG) and the PCTE Interface Management Board (PIMB). NAPI will not compete with any vendors but will provide benefits to tool and repository vendors by stimulating demand for PCTE-compliant products.

7.6.3 Goals for NAPI

NAPI has five primary goals:

- *To promote the use of the PCTE specification as the definition of a set of services for ISEEs.* NAPI will create a forum for discussing PCTE; provide newsletters and electronic bulletin boards; and invite framework vendors, tool developers, and end users to various symposia, workshops, and meetings. These activities

will inform the North American market about the use, implementation, and evolution of PCTE and the benefits of using or developing PCTE-based products.

- *To provide a recognized forum in North America for the maintenance and evolution of the PCTE standard, with the goal of becoming the United States Technical Advisory Group (USTAG) for PCTE.* This forum will develop goals for modifying or extending PCTE and will examine ways to develop PCTE to achieve those goals by working with TC33, PIMB, ISO, and other organizations. Such extensions would include support for fine-grained data, trusted systems, and object-oriented methods. NAPI will also include discussions of ISO standardization with members of ECMA and will support adoption of PCTE as a standard in the United States.

- *To liaise with other organizations to encourage the development of additional framework services required or useful for SEEs compatible with PCTE.* An integrated SEE will need additional framework services beyond those ECMA-149 currently specifies in PCTE. Incorporating additional services or modifying PCTE to ensure compatibility and efficient execution with such interfaces or standards is a goal of NAPI members. Possible candidate services for inclusion into an integrated SEE framework are as follows:

  - Services for communication, such as OMG's Common Object Request Broker Architecture (CORBA) or Hewlett-Packard's (HP's) Broadcast Message Server (BMS)

  - Additional data services such as those provided by ATIS or X3H4 (Information Resource Directory System [IRDS])

  - Services for Graphical User Interfaces (GUIs) such as the Massachusetts Institute of Technology consortium's XWindow System

  - Services for process management activities.

- *To encourage development of useful integration conventions (i.e., schemas, conventions, protocols) for integrated SEEs.* Providing a common repository is not sufficient for tools to share data. The semantics of the shared data must be agreed to in advance, and common notations and conventions must be established for sharing such information.

- *To encourage the development of a market for PCTE tools and environments.* Industry and government commitments to purchase PCTE products will

stimulate investment in the development of such products by demonstrating to developers that a market exists for PCTE products.

To achieve these goals, NAPI will undertake the following tasks in the near term.

- Encourage all members of the PCTE community to participate in evolving the PCTE standard to meet current and future needs of the software development community.

- Ensure similarity of PCTE implementations by encouraging the development of a conformance test suite. The proposed model for this process is similar to the POSIX product validation process. After the test suite has been developed, NAPI, through NIST, could certify various laboratories in North America and possibly in Europe to perform official testing to certify products as compliant with PCTE. The validation and test suite would be publicly available, however, for use by vendors and developers in performing their own in-house unofficial testing.

- Develop clear definitions of the relationship between PCTE and other framework services. Use of the NIST/ECMA framework reference model provides a basis and a consistent notation for describing framework services such as those provided by PCTE and other related standards.

- Promote the use and analysis of PCTE in universities and research institutions. Universities are a primary source of programmer expertise in such technology and for understanding and developing extensions to the standard.

7.6.4 NAPI's Organization

All NAPI contributing participants from government, industry, and academia will be partners with a shared strategy and a shared set of technical objectives decided by a consensus process. The strengths each organization brings to the initiative will shape its role in the initiative.

NAPI will consist of an executive committee of active contributors drawn from the PCTE community and four technical committees. The NAPI membership will determine the future direction and goals of NAPI; however, NAPI will be interested in listening to any comments on how PCTE should evolve.

The four technical committees and their roles are as follows:

- Technical Committee 1 (TC1) will focus on maintenance and evolution of the PCTE standard and will examine ways to develop the PCTE standard to meet

the ISEE needs of the software development community. All interested members of the PCTE community will be encouraged to contribute to TC1 because standards evolution should be an open, consensus-driven process. TC1 will make recommendations to the steering committee, which will make recommendations to TC33, ECMA, ISO, and other standards organizations. These recommendations will represent the North American consensus position on the direction PCTE should take.

- Technical Committee 2 (TC2) will be responsible for development of the validation and testing technology, including defining the scope of the test suite and interim goals for production of the test suite and directing work in this area. The test suite will be publicly available. TC2 will work with DOD and other government and industry contributors to fashion a test suite suitable for compliance testing of PCTE. TC2 will also work with the European Commission's PCTE test suite contractor so that the two test suites will be complementary. NIST will establish the mechanisms for joint development of the validation and test suite and will use the development of the POSIX test suite as a model. Technical contributions, such as PCTE vendor's in-house testing technology, could be made available to NAPI under Cooperative Research and Development Agreements (CRADAs) with NIST. These agreements would protect vendor ownership unless that code became part of the adopted validation and test suite, at which time vendors would have to give up such ownership.

- Technical Committee 3 (TC3) will promote the use and analysis of PCTE in universities. As a means of achieving its goals of promoting and evolving PCTE, NAPI supports the acquisition and analysis of PCTE by universities. The active use of PCTE at universities means that more students will get experience with PCTE, and more research in extending the standard will occur.

- Technical Committee 4 (TC4) will develop clear definitions of the relationship between PCTE and other SEE framework services to promote better integration among SEE products. TC4 will provide a forum for those interested in discussing integration issues and recommending interfaces between PCTE and other SEE products.

The first two committees are the initial NAPI technical committees. The second two are proposed additional committees to address other NAPI goals. All of NAPI's major activities will be joint undertakings by government, industry, and academia. One major role for the U.S. Federal Government will be to provide core funding, and a major role for industry will be to provide technical expertise. Therefore, no

NAPI-sponsored projects will be "government-only" projects, nor will they be proprietary and commercial.

7.6.5 Benefits of the Initiative

The initiative will benefit all potential ISEE developers and users: government agencies, vendors of software environment frameworks, tool developers, and tool users.

The U.S. Federal Government, a major user of tools and environments, benefits from the adoption of ISEE technology, and PCTE in particular, through the productivity gained by greater interoperability of tools and services. NAPI furthers these objectives by providing a forum for government agencies and other customers to discuss their needs in the evolution of the standard. A validation and testing mechanism will also assure PCTE buyers that they are obtaining a version of PCTE that complies with the specifications. Validating PCTE implementations is important to agencies that expect to acquire an accurate version of PCTE and have stated procurement goals of obtaining PCTE.

For framework vendors who are implementing PCTE, NAPI has strategic benefits. It provides a legal umbrella for exchange of technical information. Framework vendors can discuss the technical problems of extending PCTE and validating the technology and can hear what customers are looking for in PCTE products.

NAPI will enable framework vendors who are not implementing PCTE to keep informed about PCTE and discuss compatibility issues. Other framework vendors need to know what PCTE's strengths are and how PCTE compares to and interfaces with other SEE products.

Participation in NAPI will allow tool developers to discuss issues in ISEE technology and open systems and the way PCTE and other standard services should evolve to address those issues.

NAPI's use of PCTE as a baseline for ISEE technology and the U.S. Federal Government's support for PCTE products will encourage developers to create more PCTE-compliant products. This encouragement will increase competition. Several U.S. Government organizations have already announced that they intend to support PCTE. For example, the DOD's RFP for I-CASE requires that the final version of I-CASE be PCTE compliant. As PCTE evolves, the number of mandates for the use of PCTE in U.S. Federal Government procurements will increase.

7.7 PORTABLE COMMON INTERFACE SET

The PCIS Program is a North Atlantic Treaty Organization (NATO) effort sponsored by the Special Working Group on Ada Programming Support Environments (SWG on Ada PSEs). This program will define framework-level services for an ISEE. These framework services will be based on requirements identified in the International Requirements and Design Criteria (IRAC) document, and they will reflect the ECMA/NIST SEE Frameworks Reference Model. Services in the following areas will be considered:

- Object management
- Process management
- Communication
- User interface services.

The PCIS framework interface will be based on the ECMA PCTE to which a standard Ada binding exists as ECMA Standard 162. The PCIS Program will develop the framework services in the French Entreprise II environment. Plans are to complete the PCIS framework interface definition by the end of 1993. A prototype and Ada PSE demonstrator are planned for early 1994. Production quality environments based on PCIS are expected by the end of 1995.

7.8 SOFTWARE ENGINEERING INSTITUTE

The SEI is a federally funded research and development center sponsored by DOD through the Advanced Research Projects Agency (ARPA) (formerly DARPA). The Air Force Systems Command (Electronic Systems Division) awarded the SEI contract to Carnegie-Mellon University (CMU) in December 1984. In December 1989, the contract was renewed for another 5 years.

The SEI mission is to provide leadership in advancing the state of the practice of software engineering and to improve the quality of systems that depend on software. The SEI expects to accomplish its mission by promoting the evolution of software engineering from an ad hoc, labor-intensive activity, to a discipline that is well managed and supported by technology. The SEI carries out its mission by offering products and services that help SEI customers to improve the quality of their software, as described below.

7.8.1 Software Development Process

The SEI concentrates on improving the software development process. In projects related to process, SEI is assessing the actual practice of software engineering in the defense community, is training organizations to gain management control over their software development processes, and is supporting the use of quantitative methods

for software process management. Included in this focus area are the following projects:

- The Software Process Measurement Project advocates the use of measurement in managing, acquiring, and supporting software systems. The project is formulating reliable measures of the software development process and products to guide and evaluate development. To expedite DOD and industry transition, the project is actively working with professionals from Government, industry, and academia to encourage organizations to use quantitative methods to improve their software processes.

- The objectives of the Software Process Definition (SPD) Project are to establish as standard software engineering practice the use of defined processes for the management and development of software and to advance the capabilities required to define the software process within an organization. The SPD Project supports process improvement through the development and maturation of methods and technology that support process definition.

- The Capability Maturity Model (CMM) Project maintains a model describing how organizations can improve their software process maturity. This model will be continuously updated as the state of the art evolves in areas such as software engineering and Total Quality Management (TQM). It will elaborate on software development practices that provide clear strategies for capability maturity growth and improvement.

- The Empirical Methods Project develops, evaluates, and validates products (e.g., questionnaires and tests, methods and models) for use in baselining and measuring software process improvement.

- The Software Process Assessment (SPA) Project helps organizations improve their software development process by providing a structured method for assessing their current practice. It also is continuously improving the assessment method and ensuring that it focuses on organizational process improvement. The objectives of the assessment method are to identify key areas for improvement, using the SEI process maturity model as a framework, and to help the organization initiate those improvements.

- The Software Capability Evaluation (SCE) Project helps DOD acquisition organizations evaluate the capability of contractors to develop and maintain software competently. The project is improving and implementing an evaluation method that examines the software process of contractors for use in software-intensive acquisitions.

7.8.2 Software Risk Management

The SEI is exploring existing techniques and developing methods for managing risk, assessing practice, preparing organizations to manage risk, and conducting prototype risk assessment methods. To achieve its goals and objectives, the SEI must provide not only the mechanisms for managing risk but also a process that can be implemented within a project and organization to facilitate the communication of risk issues. Communicating risk underlies the strategy of addressing risk throughout the acquisition process and strengthening the relationship between Government and industry. The risk focus area was reorganized in July 1992 into the following three projects to emphasize its customer relationships:

- *The Government Risk Management Project* is the primary interface between Government customers with respect to risk management, and it establishes collaborative partnerships for developing risk management methods. Project staff develop and conduct interviews, risk assessments, risk assessment training, and risk profiles. Risk management methods are improved through active field work with Government and industry defense programs. The project is developing methods with primary framework of the acquisition life cycle. The project will develop methods to facilitate and strengthen risk communication through a rational, visible structure for identifying and analyzing risk. This project is concerned with creating viable methods for communicating risks internally within programs, which includes the communication of risk between the Government and the contractor and externally to higher levels of management.

- The goal of the *Industrial Risk Management Project* is to develop, demonstrate and transition risk management processes and techniques to an industrial client base. The project intends to fulfill its mission by working with industry partners to demonstrate methods of risk identification—the first step in risk management—and then to develop the succeeding risk management steps with a small number of strategic industry partners who are likely to be successful in transitioning software risk management into wide use on their projects.

- *The Risk Taxonomy Project* is refining the taxonomy-based questionnaire so that it can help identify software technical risk and can be easily used by development organizations. The strategy of the Taxonomy Project is to derive a software risk taxonomy by analyzing risk assessment data and other related literature, field test the taxonomy-based questionnaire, and modify the questionnaire based upon field test data. The Risk Taxonomy Project also is developing analytical methods to qualify the risks identified by the taxonomy-based questionnaire.

7.8.3 Real-Time Distributed Systems

The goal of improving the development of real-time distributed systems is achieved by integrating software engineering with systems engineering and reducing the risk associated with new technology. Projects in this focus area include the following:

- *The Rate Monotonic Analysis (RMA) for Real-Time Systems Project* aims to ensure that RMA and scheduling algorithms become part of the standard practice for designing, building, troubleshooting, and maintaining real-time systems. RMA helps engineers understand and predict the timing behavior of hard real-time systems to a degree not previously possible.

- *The Real-Time Embedded Systems Testbed Project* collects, classifies, generates, and disseminates information about Ada performance in hard real-time embedded systems.

- The goals of the *Real-Time Simulators Project* are to extend, validate, and document flight simulator and other real-time simulator architectures in a form accessible to practitioners and acquisition personnel and to understand and codify the relationship between nonfunctional quality goals and simulator software architectures.

- *The Fault Tolerance Project* is investigating the use of fault tolerance in the design and implementation of dependable critical systems.

- *The Transition Models Project* is developing a set of methods and supporting materials such as guidelines and checklists for planning, implementing, and assessing transition activities. These materials will be used by software technology producers and consumers both inside and outside the SEI. Project members also provide other SEI staff, including management, with education and training on technology transition concepts and approaches. Additionally, project members provide limited consulting on software technology transition to members of the SEI constituencies, and maintain contact with researchers and others interested in technology transition from business and academic domains.

- The objective of the *Zero-Defect Application Kernel Project* is to develop and transition software fault tolerance methodology for real-time mission-critical systems. Project members are generalizing the rate monotonic scheduling theory and developing software fault tolerance methods using redundancy. The project will combine them into a unified software engineering framework for practitioners who must meet both real-time and fault tolerance requirements.

7.8.4 Software Engineering Techniques

SEI activities related to software engineering techniques aim to increase the use of engineering knowledge for effective and efficient production of large software-intensive systems through a model-based software engineering approach and engineered project support environments. The projects in this area have been integrated around a common technical vision and strategy:

- *The CASE Technology Project and the Software Architectures Engineering (SAE) Project* were consolidated into the CASE Environments Project to address issues of engineering of environments.

- *The Domain Analysis and the SAE Projects* were consolidated into the Application of Software Models Project to address the systematic creation and application of models in application engineering.

- *The Advanced Video Technology for Imaging Project, the Requirements Engineering Project, and the Software Architecture Design Principles Project* were consolidated into the Software Engineering Information Modeling Project to address issues of capturing, representing, and making accessible increasing amounts of engineering information ranging from requirements to engineering knowledge typically found in handbooks.

7.8.5 Special Projects

The SEI is also involved in special projects. For example, the Process Research Project investigates the factors that limit software development performance by conducting research on the use of software process principles by individuals and small teams. This research is seeking insight into the processes, tools, and methods that will be most helpful in improving the performance of software engineering professionals.

7.8.6 SEI Products

With the goal of helping end users help themselves, the SEI Products group works with other groups in the SEI to develop an integrated set of products and services for managers, practitioners, and educators. SEI Products ensures that the results of SEI work are in a form the software community can easily and effectively use to improve software practice and educators can use to improve software engineering. SEI Products has the following projects:

- *The Academic Education Project* focuses on the long-term development of a highly qualified work force. The project promotes and accelerates the development of software engineering as an academic discipline. The project is developing model curricula and promoting the establishment and growth of

software engineering programs, as well as working to increase the amount of software engineering content in computer science programs. The project produces educational materials that support the teaching of software engineering in universities.

- *The Continuing Education Project* interacts with industry and Government to increase the availability of high-quality educational opportunities for software practitioners and executives. The project produces the Continuing Education Series and the Technology Series. The Continuing Education Series provides video-based courses designed for clients' in-house education, and execu offerings designed for decision makers involved in improvement efforts. Technology Series provides stand-alone presentations that promote awareness of emerging issues and leading-edge technologies.

- CMU offers a *Master's in Software Engineering (MSE) Project*, a 16-month master's degree program in software engineering in response to industry's growing demand for skilled software developers. The program is a cooperative effort of the CMU School of Computer Science and the SEI. The core of the program is based on the SEI curriculum recommendations for MSE programs. The MSE Project also produces the Academic Series, a set of video-based graduate-level courses on software engineering.

7.8.7 SEI Services

SEI Services works with other groups in the SEI to develop, deliver, and transition services that support the efforts of SEI clients to improve their ability to define, develop, maintain, and operate software-intensive systems. To accelerate the widespread adoption of effective software practices, SEI Services works with client organizations that are influential leaders in the software community, promotes the development of infrastructures that support the adoption of improved practices, and transitions capabilities to Government and commercial associates for use with their client organizations. SEI Services is composed of the following groups and functions:

- *The Computer Emergency Response Team (CERT)* was formed by ARPA in November 1988 in response to the needs exhibited during the Internet worm incident. The CERT charter is to work with the Internet community to facilitate its response to computer security events involving Internet hosts, to take proactive steps to raise the community's awareness of computer security issues, and to conduct research targeted at improving the security of existing systems.

- *The Improvement Planning and Organizing (IPO) Function* focuses its activities on SEI clients who seek long-term support for their software process

improvement efforts. IPO was formed to address needs for integrated software process improvement programs. These include understanding the principles of how to effectively launch and sustain continuous software process improvement and integrating assessments, organizational dynamics, the maturity model, process definitions, and improvement metrics into a plan. IPO members provide support in planning and organizing continuous software process improvement programs. They do this by using business and case histories in software process improvement to illustrate benefits achieved, by promoting and launching software process improvement programs, and by coordinating a client's activities with the work of different SEI projects.

- *The Organization Capability Development Function* supports clients' software process improvement efforts by helping the client organizations develop the capability to manage the organizational aspects of improvement at their sites. Services include organizational assessment, vision setting and dissemination, strategic planning, transition infrastructure development, executive consulting, cross-functional team development, and management of technological change, and provision of consultation for software engineering process groups. The goal of the function is to provide to clients the self-sustaining capability of managing their own long-term improvement.

- The strategy of the *Technical Assistance Project* is to define, develop, and implement a structured technology-transition process that will establish the requisite technology-transition capabilities. This process will enable software technology to be disseminated broadly. Applying a structured transition approach will accelerate the transition and adoption of improved software engineering practices and technology.

7.9 SOFTWARE EXECUTIVE OFFICIAL COUNCIL

The DON Software Executive Official Council (SEOC) will serve as an advisory committee that focuses on software-related technology and policy issues. The council will address software issues that affect AIS, C3I systems, and embedded weapon system software. The council will meet quarterly with Flag and Senior Executive Service (SES)-level representatives from the Chief of Naval Operations (CNO), Commandant of the Marine Corps (CMC), major DON Systems Commands (SYSCOMs), Program Executive Offices (PEOs), and Navy research centers and laboratories.

7.10 SOFTWARE TECHNOLOGY FOR ADAPTABLE, RELIABLE SYSTEMS

STARS is a technology development, integration, and transition program to demonstrate a process-driven, domain-specific, reuse-based approach to software

engineering (also known as megaprogramming) supported by appropriate tools and environment technology.

The goal of the STARS program is to increase software productivity, reliability, and quality by synergistically integrating support for modern software development processes and modern reuse concepts into the latest SEE technology. To meet that goal, STARS has set the following objectives for the 1992–95 time frame.

- *Software Reuse.* Establish the basis for a paradigm shift to reuse-based development
- *Processes.* Establish capabilities for tailoring process definition and management
- *Environments.* Establish adaptable, commercially viable SEE solutions that are available on multiple vendors' platforms, are built upon open architecture industry standards, and include automated support for process management and software reuse
- *Demonstration and Validation.* Demonstrate that the STARS integrated reuse, process, and SEE solutions can be used in actual practice to increase the quality and life-cycle supportability of DOD software products
- *Technology Transition.* Sponsor activities and disseminate information that will speed up transition of STARS technologies to practical use.

STARS technical development addresses new areas. Therefore, STARS program management selected three prime contractors—Boeing, IBM, and Paramax—to reduce risk and accelerate acceptance of changing technology. Combining the efforts of three prime contractors will enable cooperative work to be accomplished from a very broad experience base. Furthermore, use of multiple prime contractors and their subcontractors will help accelerate the shift to megaprogramming in other companies.

7.10.1 Reuse

STARS is working to establish a basis for a paradigm shift to reuse-based development. As part of the activities, technical, management, cultural, and acquisition-related issues are being considered with the goal of reducing the adoption risks in transitioning to reuse-based software engineering. STARS reuse activities include establishing a framework for reuse processes, providing automated support for key processes, and experimenting in the definition and prototyping of reuse library open architectures. The STARS reuse approach focuses on an iterative model

that addresses technology evolution and cultural issues with a trial-usage and feedback loop. STARS technology transition affiliates provide feedback that is incorporated into the concept, processes, guidelines, and automated tools. The planned reuse results include reuse-transition support guidelines; a reuse-based concepts document; modular descriptions of reuse processes associated with various user roles; a reuse library open architecture framework; reuse library mechanisms that support acquisition, classification, browsing, and retrieval; general management of reusable assets; and additional tools to support the various reuse processes.

7.10.2 Process

STARS will establish capabilities for process definition and management that will show the value of process concepts, process definition, process tailoring, and process support in the environment as a vehicle to improve quality, productivity, and reliability. Process definition and tailoring capabilities will support the SEI CMM. STARS process technology transition affiliates will provide feedback to improve the processes and products.

7.10.3 Environment

STARS is working with framework providers, tool vendors, and standards organizations to ensure that commercially viable environment infrastructures (frameworks) are extensible and robust and conform to open architecture standards. Framework-based environments serve as integration platforms on which tools, services, and functional capabilities can be integrated to support software development within the context of megaprogramming.

7.10.4 Demonstration

To measure the success of STARS technologies, three demonstration projects have been selected that will use STARS technologies to develop operational mission-critical applications in Ada:

- A 200,000 line-of-code subsystem of the Improved Guardrail V system will be reengineered by an in-house support contractor at the Life-Cycle Support Center at the Army Communications Electronics Command (CECOM), Ft. Monmouth, New Jersey. (Application domain: Electronic Warfare)

- A 200,000 line-of-code subsystem of the Space Warning Mission will be reengineered with contractor support at Air Force Space Command (AFSPACECOM), Colorado Springs, Colorado. (Application domain: Command and Control)

- A 110,000 line-of-code system at the T-34C Flight Instrument Trainer will be developed at the Navy Training and Simulation Center (NTSC), Orlando, Florida. (Application domain: Flight Trainers)

The projects will be designed to provide a pragmatic measure of the progress STARS has made in developing and introducing new software engineering approaches and to provide realistic and useful feedback to the technology developers. Advanced planning is under way, and execution of these projects is scheduled to begin in late 1993.

7.10.5 Technology Transition

STARS is developing an overall technology transition strategy that identifies specific activities to foster the transition of the STARS concept and technologies to practical use. Technology partnerships have been formed with potential customers and suppliers of STARS technologies using the STARS Technology Transition and Prime Affiliates Program as well as relationships between the STARS prime contractors and their commercial counterparts. General information about STARS concepts and technologies is disseminated through newsletters, participation in selected software conferences, electronic bulletin boards, and the annual STARS conference.

7.11 TAC-4 AND TAC-5 PROCUREMENTS

The Secretary of the Navy, the Office of the Secretary of Defense (OSD), and the General Services Administration (GSA) have approved the TAC-4 contract to procure the latest high-performance, open systems-compliant (insofar as current standards allow) computers in support of tactical, strategic, business, and administrative functions. The contract is a 3-year indefinite delivery, indefinite quantity, fixed-price ordering contract that provides for 3 additional option years for maintenance.

TAC-4 is managed by the Naval Command, Control, and Ocean Surveillance Center (NCCOSC) in San Diego. Designed for Navy afloat applications, TAC-4 is also available to other DOD components, other U.S. Government agencies, and foreign nations aligned with the United States. The Army, Air Force, Coast Guard, and Marine Corps have already presented their requirements in connection with the TAC-4 contract.

TAC-4 provides a suite of equipment that is binary compatible and required to interface downward with equipment provided by the previous TAC-3 contract. Both ruggedized suites (i.e., standard 19-inch, rack-mounted, environmentally hardened), and purely commercial suites will be offered. The TAC-4 contract provides

performance incentives for the high-end equipment. The TAC-4 will be procured with an Ada capability. The requirement is based on the stated current and near-term needs of the user.

The TAC-4 solicitation is a prime example of the DON's riding the price and performance curve on hardware, achieving a common operating environment, extending it beyond command and control to other forms of communications and nontactical applications, and doing what the strategy of buying off the shelf is intended to do.

The TAC-5 contract is scheduled to follow approximately 24 months after the TAC-4 award (i.e., in May 1996). The purpose of the rapid turnaround is to ensure that the Navy maintains currency with evolving technology and benefits from the attendant reductions of cost per unit of performance.

7.12 PLANS

Currently, several DOD planning initiatives relating to Ada and Ada-related technologies are in process. The subsections below provide synopses of these activities.

7.12.1 Software Action Plan

The Acting Director of Defense Research and Engineering (DDR&E) established the Software Action Plan Working Group (SWAP-WG) in June 1991 to specify and implement ". . . an integrated technology and management plan to ensure more cost- effective support of weapons systems and related test equipment systems within DDR&E's purview." The SWAP-WG directly supports the SEOC, which is chaired by the DDR&E.

To accomplish its mission, the SWAP-WG has addressed high-leverage software management and technology issues to support four basic goals:

- Assist the DOD in establishing a proactive acquisition and life-cycle management process
- Identify and act upon opportunities for improving DOD software policies, standards, and guidance
- Identify opportunities to strengthen the capabilities of the DOD software work force

- Support and capitalize on current software technology programs and promote the integration of the resultant products into other SWAP-WG- and DOD-SEOC-sponsored efforts.

Specific efforts in which the SWAP-WG has thus far provided funding and/or technical support to help achieve those goals include the following:

- Software process improvement
- Assessment of the maturity or capability of software acquisition organizations
- Software risk assessment
- Core set of software metrics
- Software reengineering
- High-level language policy
- Software life-cycle standards
- Software cost reporting standards
- Standards-based architectures for weapon system software
- Software engineering environment standards
- Software education for DOD senior executives
- Enhancements to the software personnel base
- Enhancements to the DOD software technology base.

The SWAP-WG has been successful in leveraging the expertise of its membership and the limited resources available to it and has served as an effective mechanism for addressing numerous DOD software-related issues. DDR&E's and SEOC's continued support of the SWAP-WG's efforts will enable the entire Department to reap additional benefits in addressing the many managerial and technical challenges presented by DOD's increased reliance on software.

7.12.2 Draft DOD Software Technology Strategy Document

In December 1991, a Draft DOD Software Technology Strategy Document was completed and distributed for public review and comment. This document justified a coordinated set of DOD software science and technology actions and investments that would meet DOD needs for improved software functionality and that would bring future DOD software costs under control. It identified the following two levels of software technology investments:

- A current program that could be implemented within currently programmed budget levels

- An achievable program that focuses on more cost-effective levels of software technology that could be realized with higher levels of funding.

The document covers a 5-year period of DOD software technology investments between FY92 and FY07. It provides more detail for the first 5 years.

The Draft DOD Software Technology Strategy Document was used as the baseline for a new DOD Software Technology Initiative (STI) program that will begin in FY94. The STI program represents a focused and integrated initiative to accomplish the following:

- Provide significant and timely boosts to DOD thrust areas through experimental use of software technologies on advanced technology demonstrations

- Provide support for new and existing systems by addressing current voids in the DOD software science and technology program.

The military departments and ARPA will plan and execute the STI program. The DDR&E retains approval authority for the plan and its subsequent execution.

7.12.3 DON Reuse Implementation Plan and Guide

The DOD estimates that expenditures for developing and maintaining software for its weapons, command and control, and other automated information systems currently exceed $24 billion a year. In an attempt to better manage these costs and improve its ability to develop and maintain high-quality software, DOD has initiated a comprehensive effort to incorporate software reuse practices into its software development efforts.

DON has developed the Draft DON Software Reuse Implementation Plan to establish a reuse infrastructure in the DON. This plan will establish systematic and structured software reuse as an integral part of the DON software-systems development and acquisition process. DON believes systematic and structured reuse can help decrease the cost of software acquisition and development. In addition, the DON believes that an effective reuse infrastructure will improve software reliability, productivity, portability, and interoperability.

Planned initiatives include the following:

- Establish DON-wide reuse policies, standards, and guidance.

- Establish a Reuse Manager under the CNO to provide oversight on the DON reuse initiative.

- Establish Domain Managers to manage the domain engineering activity within each PEO and SYSCOM activity.

- Establish Reuse Coordinators to identify sources within PEOs and SYSCOMs and to share reusable components.

- Develop incentive programs to produce long-term cost savings from effective reuse for all levels of DON, to include PEOs, SYSCOMs, and Direct Reporting Program Managers (DRPMs).

- Establish a DON Reuse Executive Council (a function of the DON SEOC) to develop reuse policies, standards, and guidance.

- Establish an education and training program targeting the DON software reuse concepts for senior executives.

- Establish a Domain Analysis Pilot Project under the CNO to use as a basis to build a DON reuse infrastructure.

7.12.4 DON Information Management Strategic Plan

Pursuant to Defense Management Review Decision (DMRD) 918, the DON is turning most Navy and Marine Corps Central Design Activities (CDAs) and Data Processing Installations (DPIs) over to the Defense Information Systems Agency (DISA) for management. In the future, DON will be receiving software development and operations support from DISA on a cost-reimbursable, fee-for-service basis. To accommodate this change and to focus increased attention on the information management functions that remain with DON, the Naval Information Systems Management Center (NISMC) has initiated DON-wide strategic planning for information and computer resources. Emphasis over the next year will be on the following:

- Developing a common naval shipboard hardware and software architecture for Navy and Marine Corps afloat and amphibious tactical support and combat service support functions. This architecture for the Naval Tactical Combat Support System (NTCSS) will permit integration of tactical and tactical-support of the Naval Expeditionary Forces under the new naval maritime strategy.

- Developing Service-level agreements for support to DON activities from DISA-managed CDAs and DPIs and institutionalizing unit costing and a fee-for-service structure. DON expects to generate new cost savings by taking advantage of lower rate structures generated through economies of scale.

- Improving base-level computing and communications support capabilities by migrating from the current heterogeneous sole-source environment toward a clie- -server architecture that is open-systems compliant, most cost-effective, anc :ss personnel intensive. Standard acquisition vehicles will be used, es- :ially ongoing DISA Indefinite Quantity contracts and Basic Ordering Agreements.

- Optimizing the Return On Investment (ROI) for all DON information technology expenditures by putting into place the FPI management process initiated by OSD under CIM and institutionalizing the requirement for Functional Economic Analysis as justification for information technology acquisitions.

- Making DON information practices more effective by continuing or initiating the following management improvements:

  - Participating actively in DOD data element standardization efforts

  - Focusing life-cycle management attention on economic issues (e.g., ROI and real cost savings) and bringing the expertise of the Naval Center for Cost Analysis (NCA) to bear at key decision points

  - Developing a standard software engineering environment through a combination of initiatives such as the following:
    I-CASE Pilot Program
    Ada Implementation Guide
    STARS/ARPA Demonstration Project
    Reuse Implementation Plan and Guide
    DON SEOC
    Multilevel Information Security

  - Developing a standard DON information technology technical architecture in conformance with the Defense Information Infrastructure Technical Architecture For Information Management (TAFIM).

7.12.5 Software Process Improvement Plan

The SPI Plan is based on the SEI's CMM, which is a five-level framework of key process areas. This framework characterizes the maturity that organizations use to establish or improve their software processes. It is assumed that the maturity level of development and maintenance organizations directly correlates to their development capabilities. DON's goal is to develop a program where all software

development and maintenance organizations undertake self-improvement programs to raise their maturity levels. Initiatives include the following:

- Develop a DON-wide SPI implementation plan.
- Develop a training program to educate or enhance management understanding of the effects of software on acquisition, development, and post-deployment software support strategies.
- Enhance the Software Process Advisory Group to help institutionalize continuous process improvement through the sharing of lessons learned.

The SPI Implementation Plan calls for the use of an SCE as part of the source selection process for all new programs started in FY94 through FY96.

The Director of Defense Information, as part of an Information Technology Policy Board (ITPB) initiative, requested volunteers from the Services to participate in an SPA. The Navy Fleet Material Support Office (FMSO) and Naval Research and Development (NRaD), a division of the Naval Command, Control, and Ocean Surveillance Center, volunteered to participate in this program. The FMSO assessment has been completed; the NRaD assessments began in August 1993.

The most important benefit FMSO gained from this assessment was the intensified focus on the need for formal software process improvement. The assessment provided the process for formally identifying key areas throughout the organization that needed improvement and a baseline on which to measure future improvement. The resulting action plan will now allow FMSO to rank improvement efforts based upon need and funding availability.

7.13 DON TECHNOLOGY PILOT PROJECTS

The DON is involved in several pilot projects that will implement many of the software engineering principles and Ada features discussed in this guide in real-world situations. The sections below briefly describe these projects.

7.13.1 Integrated Computer-Aided Software Engineering Pilot Project

The I-CASE procurement, as discussed in Section 7.4, is a major DOD initiative, led by the Air Force, to procure an Ada-based integrated computer engineering environment and associated services, something software developers in DOD have been lacking for a long time.

The Director of Defense Information requested nominations for pilot projects from the Services, and four of the nominated DON pilot projects were selected. These projects, which will begin immediately after contract award, are as follows:

- *Marine Air/Ground Task Force Logistics Automated Information System.* This mid-size pilot project proposes to conduct both reengineering and reverse engineering actions on the seven systems that support mobilization and deployment of various Fleet Marine Force Units.

- *Communication Support System Standard Communication Environment.* This project addresses the redesign and integration of the configuration control, status monitoring, and performance monitoring modules of the real-time communication management requirements for Navy tactical communications, including satellite and line-of-sight, high-frequency radio systems.

- *Navy Tactical Command Systems Afloat.* By using rapid prototyping techniques, this pilot project will develop an automated reasoning module for analyzing combat information related to engagement analysis, commander's estimates, effects on sensors and countermeasures, and decoy diffusion or drift. This module also will include a decision support system to assist tactical commanders in effectively using critical elements of information associated with hard- or soft-kill actions.

- *Automation of Procurement and Accounting Data Entry.* This project consists of reverse engineering that uses Ada on one of eight subsystems that support monitoring and management of the procurement process. This system is currently in COBOL.

The successful introduction of I-CASE into DOD in the future will depend greatly on the outcome of the pilot projects established by the DDI.

7.13.2 Functional Process Improvement

Early in 1992, DON initiated a program to assess, through four pilot projects, the FPI methodology and management process proposed by OSD for process improvement. The four pilot projects were:

- Intermediate maintenance across the aviation, surface, and submarine environments in the DON

- Pay and personnel data collection in the Personnel/Support Activities and Personnel Support Detachments

- Funds allocation in the Weapons Division at the Naval Warfare Development Center

- Training request processing at Naval Sea Systems Command (NAVSEA) Human Resources Offices.

Included in the pilots was an assessment of the use of an Integrated System Definition Language (or IDEF) for process and data modeling. Pilot project managers provided final briefings on 3 March 1993.

The lessons learned from the pilot projects and recommended guidelines for future Project Managers are being published in a DON FPI Implementation Guide.

7.13.3 SEI Pilots

DON is working closely with SEI to transition SEI's technology into DON projects. Some of the following projects are included in this effort:

- Use of RMA in the software design of the AN/BSY-2, DON's largest Ada project, which consists of 2.4 million lines of unique Ada code. Other modules reuse 1.2 million lines of code within the BSY-2.

- Transition of fault tolerance technology to the Office of Naval Research.

- Transition of RMA principles in standards for NGCR operating systems and LANs.

- Transition of a measurement program for improving the software process at the Naval Air Warfare Center.

- Development of software architectural components for the AN/SSQ-94 trainer.

- Methods and processes for managing and communicating software risks for PEO for Air ASW, Assault, and Special Mission Programs.

- Transition of visual imaging technology for converting hard-copy Naval Supply Systems Command (NAVSUP) engineering drawings to computer-aided design.

- Course material development for Space and Naval Warfare Systems Command (SPAWAR) OSA training.

7.13.4 STARS Demonstration Pilots

A Memorandum of Agreement was signed between ARPA and NAVAIR in October 1992 for a joint project to apply the STARS megaprogramming paradigm to the rehost upgrade of the Navy's T-34C FIT. The development is expected to start in January 1994 and to be completed in October 1994. Objectives of the project are as follows:

- To build a real software-intensive product by using a process-driven, domain-specific, reuse-based, and technology-supported approach (megaprogramming)
- To measure the benefits of megaprogramming and provide feedback
- To transition the demonstration organizations to megaprogramming.

This demonstration project is part of a SHOW ME philosophy. If successful, it will encourage other DON projects to transition to megaprogramming.

Section 8
Training and Education

This section provides practical information on implementing an Ada training and education program. Topics include (1) organizational training requirements, (2) training and information sources, and (3) lessons learned and recommendations.

The successful transition of new technology into an organization is directly related to the effectiveness of the training and education program that introduces it. Any education and training program must focus primarily on software engineering and teach Ada in the context of a tool.

Introducing Ada within an organization demands the development of a well-planned education and training program tailored to the needs of the organization. Risk to a program is minimized if education and training are sufficiently funded and adequately planned.

8.1 ORGANIZATIONAL TRAINING REQUIREMENTS

Organizations vary widely in their structure, in the applications they use, and in their training requirements, but all organizations face similar education and training needs.

8.1.1 Course Content

Training and education are needed in the areas of Ada orientation, programming language, software engineering, development support environments, and project management. The following course topics are recommended for each subject area:

- Ada Orientation
  - History of Ada
  - Software engineering principles and the way Ada supports them
  - Pros and cons of using Ada
  - Unique features of Ada
  - Phases of the software system life cycle
  - Amount of effort associated with each phase
  - Cost associated with each phase
  - Features of Ada targeting software maintenance
  - Ada procurement strategies
  - Ada as an alternative to other languages
  - Object-Oriented Design (OOD) constructs
  - Reuse
  - Unique training aspects of Ada
  - Computer-Aided Software Engineering (CASE) tool overview

- Effects on the software life cycle of using Ada
- Importance of configuration management

- Ada Programming Language
  - Introductory Courses
    - Software engineering concepts
    - OOD
    - Libraries and reuse
    - Ada background
    - Program units
    - Packages first, then subprograms
    - Instantiation and use of generic units
    - Separate compilation
    - Early interface testing
    - Types (including access, private, and limited private types)
    - Subtypes, declarations, and statements
    - Exceptions and exception handling
    - Elaboration and execution definition/differences
    - Daily hands-on and practical workshops
    - Input/Output (I/O)
    - Tasking overview
  - Advanced Courses
    - Concurrency and the tasking model
    - Real-time programming
    - Low-level features such as representation clauses
    - Pragmas
    - Interfaces with other languages, modules, and databases
    - Creation of generic units and planning for reuse
    - Design and reusability
    - Daily hands-on and practical workshops
    - Performance profiling

- Software Engineering Using Ada
  - Software engineering concepts
  - OOD
  - Ada-specific design considerations (e.g., data typing, structures, exception handling)
  - Software standards and documentation requirements
  - System development life cycle (e.g., requirements analysis, design, implementation, testing, maintenance)
  - Design of reusable components (generics)

- Selection and use of database and file strategies
- Programming-in-the-large issues

- Ada Development Support Environment
  - Training tailored to specifics of hardware and user interface
  - Extensive on-line exercises
  - Library management and configuration control
  - Selection and use of tools (e.g., text editors, linkers or loaders)
  - Selection and use of compilers and automated tutorials

- Ada Project Management and Cost Estimating
  - Software engineering concepts
  - Software Quality Assurance (QA)
  - Configuration management (e.g., performance baselines and changes)
  - Measurement of cost, productivity, and risk
  - Prototyping (i.e., rapid versus evolutionary)
  - Resource allocation (e.g., hardware, staff, training)
  - Software sizing
  - Portability
  - Reuse
  - Sources of information.

8.1.2 Evaluation of Education and Training

In order for the Department of the Navy (DON) to successfully and cost-effectively evolve toward modern software engineering practices using the Ada programming language for all computer systems, education and training programs must be well planned and critically evaluated. The paragraphs below provide guidelines for this task.

8.1.2.1 Target Audience and Environment

Planners should define the target audience and environment for training by evaluating the categories of personnel and available facilities. Analysis should be conducted to identify the amount, degree, and level of training required for an individual. Course prerequisites should be enforced so that instructors do not digress from presentation of targeted material to remedy deficiencies. Next, the environment should be analyzed in terms of computer facilities, tools, project types, and project deadlines. The facilities and tools needed for training may be already available, or a procurement lead time may be involved. The types of projects and their deadlines should be reviewed to ensure planned training matches workload requirements and schedules. Optimally, training should begin and end immediately before actual project work is scheduled. This type of scheduling is complex and dependent on many interactive factors. Charting techniques are useful to clearly show training and project dependencies.

8.1.2.2 Timing and Course Length

The timing of education and training is key. The optimum solution is to have a fully qualified Ada manager, engineer, or programmer emerge from the training process and immediately become part of an active Ada project. The time lag between training completion and task commencement should be kept to a minimum. If delay between the two is unavoidable, provision should be made for refresher training to be available when required.

For introductory courses, 3 to 5 class days are needed to teach the basics of the language and to introduce the underlying software engineering principles. For advanced classes, 10 to 20 class days are needed to present the material and enable trainees to understand the complexity of the subject matter and become proficient with the language, compilers, and automated tools. For executive overviews and briefings, 2 to 8 hours are required to present and discuss the unique attributes and requirements of software engineering using Ada. At least 5 days should elapse between courses to allow students time to digest course material.

8.1.2.3 Testing

Courses should provide pretesting, progress testing, and post testing of student knowledge and expertise. Such tests allow courses to be tailored cost-effectively to the unique requirements of the students. Pretesting assesses the expertise and background level of the class as a whole and determines the beginning achievement level of individual students. It also helps instructors to direct course emphasis
and identify special instructional requirements. Progress testing provides the instructor and students with a measurement of progress and mastery of the subject. Post testing provides an indication of the knowledge and proficiency achieved by the student and of the overall course effectiveness. Consistently ineffective courses should be dropped from the training plan or redesigned.

8.1.2.4 Location

Depending on the number of students, on-site training is usually more cost-effective than off-site training. When possible, instructors should be brought to the site, or instructors already on-site should be used. This approach reduces travel and per diem costs for students, reduces time away from the workplace, and provides training in the actual work environment on installed Ada compilers and automated tools to be used with actual project work. As in-house expertise and knowledge increases, in-house staff should be used to update, improve, and present on-site training and to function as mentors to less experienced staff.

8.2 TRAINING AND INFORMATION SOURCES

Education and training are available from a wide variety of sources. The subsections below provide examples of available sources of training. Appendix A, Sections A.1.2 and A.3.1, presents information on accessing these sources.

8.2.1 Academic Institutions

Appendix A, Section A.3.1, lists civilian academic institutions that currently teach or use Ada.

8.2.2 DOD Organizations and DOD-Sponsored Activities

Ada training and/or information is available from the following Government sources:

- Ada Language System/Navy
- Ada Software Engineering Education and Training Team
- AdaSAGE
- Air Force Institute of Technology
- Common Ada PSE Interface Set (CAIS)
- Computer Sciences School (Marine Corps)
- Computer Science School (Army)
- National Audiovisual Center
- National Defense University
- Naval Postgraduate School
- Software Engineering Institute
- United States Air Force Academy
- United States Air Force Technical Training School
- United States Army Engineering College
- United States Military Academy
- United States Naval Academy.

Appendix A, Section A.1.2 provides information on how to access these sources.

8.2.3 Catalog of Resources for Education in Ada and Software Engineering

The Catalog of Resources for Education in Ada and Software Engineering (CREASE), produced by AdaIC, lists military, commercial, and academic sources of courses for education in Ada and software engineering. The document is available on-line and may be accessed and downloaded from the ajpo.sei.cmu.edu host, as described in Appendix A, Section A.2.

8.2.4 Other Sources of Ada Training Information

Other non-DOD sources that may provide information or pointers to information on Ada education and training are the local Special Interest Groups on Ada (SIGAdas) and *Ada Letters* published by the Association for Computing Machinery (ACM). Additionally,

the following annual conferences are good sources of Ada and software engineering information:

- ASEET Symposium
- DOD Software Technology Conference (formerly the STSC Conference)
- National Conference on Ada Technology
- SIGAda Conference
- Tri-Ada Conference
- Washington Ada Symposium.

8.3 LESSONS LEARNED AND RECOMMENDATIONS

The points listed below highlight lessons learned from DOD, industry, and academia concerning Ada as an effective software engineering tool. The most significant lesson is that management commitment is critical to success. The sources of the following lessons are indicated in parentheses at the end of each entry.

- Ada ***must*** be taught as a software engineering tool, not syntactically as yet another programming language.

  - Software engineering education is mandatory for the proper use of Ada. To accrue the desired productivity gains and cost savings, education and training in the features of Ada that support the principles of software engineering, such as modularity, abstraction, and information hiding, must be provided. A low-level syntactic focus on Ada as a language is ineffective. (SEI)

  - A study on Ada education and training, conducted by the Armed Forces Communications and Electronics Association (AFCEA), indicated that training in software engineering practices was both the most important and the least practiced of all training requirements. The AFCEA surveys were consistent on this point. General comments often were made that the high-level concepts of Ada (e.g., packages, tasking, generics, strong data typing) permitted new options in software architecture and design. The intelligent use of these concepts would have great benefit for both the software engineering process and the quality of the resulting design. (AFCEA, 87)

  - Teaching should not be done solely by analogy, (i.e., with simple transliteration of examples from other languages). Teaching by analogy alone will fail to capitalize on the power of the many new software engineering features of Ada. Resultant programs will be "AdaBOL" or "AdaTRAN" (i.e., syntactically Ada but semantically constrained by the limitations of the earlier language). Moreover, if analogy alone is used in teaching, the student is less likely to readily adopt new, more powerful ways of accomplishing old tasks

and ways more suited to Ada, and to take advantage of "new" Ada features such as packages, tasking, exceptions, aggregate assignments, and use of Boolean expressions. Students must be taught to think in Ada. (SEI, 92)

- Hands-on training is essential.

  - Ada education and training are ineffective without hands-on experience on actual Ada projects. The training can be conducted on an actual project, on a pilot study, or in classroom exercises. Several respondents to the AFCEA 87 study agreed that language training was useless without such immediate reinforcement. Actual project experience was the most highly recommended form of hands-on training. (AFCEA, 87)

  - It is also apparent that classroom training alone, even with hands-on exercises, does not prepare individuals fully for actual work on Ada projects. To be fully educated, people need to have on-the-job experience either on an actual Ada project or on a pilot project with actual deadlines and goals. (AFCEA, 87)

  - Examples must be at a real level to accelerate the ability to apply the language techniques. Students have difficulty applying new concepts to their application areas without real-life models to imitate. Ada compounds the problem by having new constructs. Thus, many programmers who have a foundation in Assembly have no parallel constructs to which to relate. "Generics" is a good example. (AFCEA, 87)

  - Additional models (i.e., analogies) need to be provided to facilitate the application of Ada to specific projects; for example, avionics examples could be created to provide real Ada coded examples to show how Ada solves problems in this specific application domain. Models are needed to allow new Ada programmers to imitate good style and practices rather than to start from scratch. (AFCEA, 87)

  - When asked about learning Ada, several interviewees responded that there is no substitute for on-the-job training. (AFCEA, 87)

  - It is imperative that hands-on programming exercises be a major component of the training process. It is just as difficult to build computer software and learn computer languages without practical applied work as it is to learn how to repair radar sets or jet engines without applying the skills. In fact, the more detailed the training, the more imperative hands-on exercises become. For example, in classes dealing with using Ada for a specific embedded

processor, there is no substitute for writing software with the compiler and tool set that will be used on the actual development program. The software engineers working at this level need to understand the performance and limitations of a particular tool set. (SEI, 92)

- Training and education and their application must be closely aligned.

  - Course documentation, materials, and models should be relevant to the actual work environment. For example, a class of Automated Information System (AIS) students should be working with sample Ada programs for unclassified business functions. Normally, instructors should follow these presentations by assigning students exercises that require use of automated compilers or other tools so that students will master the subject matter with hands-on training. The compilers, tools, and models should be the same as those used by the students in their actual work environments.

  - It is important to assign useful work in Ada immediately after training. Education provided too long before actual use is not reinforced and is lost. (AFCEA, 87)

  - In general, knowledge retention was best when the training was concurrent with or very close to actual project assignment. Training benefits were judged to be essentially lost if as few as 4 to 6 weeks passed between the training class and project assignment. (AFCEA, 87)

  - From the point of view of students, immediate need for Ada skills is the strongest motivator for learning. (AFCEA, 87)

- All personnel associated with an Ada development effort require some level of Ada education and training.

  - Training is necessary for both acquisition and development of maintenance personnel. Software designers, systems engineers, Ada programmers, and software managers in both industry and Government need to know about Ada and its associated software engineering techniques. Ada training should address all of these groups, but the depth and kind of knowledge needed will vary for each. (SEI, 92)

  - Project support personnel need training in relevant aspects of Ada. (AFCEA, 87)

- Access to an Ada software programming expert is important.

  - Student access to an Ada programming expert also was found to be both essential and effective in on-the-job training situations. Having available a recognized expert to answer questions and provide guidance to novice learners was a good catalyst for the learning process. (AFCEA, 87)

  - It is useful for a project to build a core group of knowledgeable, well-trained individuals to serve as mentors for more junior personnel. These individuals *must* be very knowledgeable in the application of Ada and software engineering and in the project domain. Experience has shown that a few such individuals, placed in key positions within the project, can have a stimulating effect on other engineers. (SEI, 92)

- CAI is good for supplementing other training but not as the sole method of providing training. Available methods and media must be compared against training funds, objectives, and desired outcome. Lectures with some visual aids are relatively inexpensive and suitable for briefings, overviews, and orientations.

- Providing adequate tools and using only validated Ada compilers for training is important.

  - Adequate, user-friendly tools greatly enhance the acceptability of Ada, particularly by professional programmers. An overtaxed or overloaded support environment often creates frustration, which is displaced to the language. For example, a system where editor response time is degraded will generate a negative attitude toward the language although the editor, not the language, is at fault.

  - Using only validated Ada compilers is recommended so that students are not exposed to the frustration of studying an Ada feature and then discovering it does not work in their training environments.

  - Although use of nonvalidated or subset compilers may seem effective or convenient in the short term, productivity on the job will suffer when students have to adjust to the full language.

- Ada is not significantly more difficult to learn or teach than are other languages. Teachers and students do not find Ada significantly more difficult to learn and teach than alternative languages, and student reaction is generally favorable. (*Communications of the Association for Computing Machinery* [CACM], November 92)

- Teaching Ada effectively requires a different methodology from those used to teach previous languages. Emphasis needs to be placed on aspects such as programming-in-the-large, team development, and maintenance. To do this well requires redesigning old lesson plans and incorporating libraries of software packages.

  - Instructors must redesign their teaching methodologies to teach Ada and its power effectively. Pascal style and methodology do not carry over to Ada. (CACM, November 92)

  - Basic language constructs in Ada take longer and are more complicated to teach than basic language constructs in Pascal. Thus, first and second programming courses require more effort from the instructor, but these extra efforts will be beneficial. (CACM, November 92)

  - Students should be exposed to "reading" and analyzing Ada software systems before "writing" or creating them.

  - Two kinds of support are envisioned for an Ada-based freshman course, which are currently unavailable. For instance, a large library of well-designed Ada library units is needed. Abstract data types are not enough. Examples are needed that illustrate specific key features of Ada, kinds of objects and operations, and other software design issues, as well as a large collection of components that students can combine into interesting systems.

References

Archer, T. S. "Managing the Ada Conversion and Integration of Mission Critical Defense Systems." 9th Annual National Conference in Ada Technologies. March 1991.

Association for Computing Machinery, *Communication*, January 1984.

Baumert, J. and M. McWhinney. *Software Measures and the Capability Maturity Model* (CMU/SEI-92-TR-25, ESC-TR-/92-025). Pittsburgh, PA: Carnegie-Mellon University, 1992.

Boehm. B.W. *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice-Hall, 1981.

Carnegie-Mellon University/Software Engineering Institute. *Software Capability Evaluation Overview*. Pittsburgh, PA: Carnegie-Mellon University, 19-21 March 1991.

Contrast: Ada 9X and C++, Schonberg, E. New York University, 1992 (Distributed by Ada IC on Form S100-0992B).

Humphrey, Watts. *Characterizing the Software Process: A Maturity Framework* (CMU/SEI-87-TR-11). Pittsburgh, PA: Carnegie-Mellon University, June 1987.

Jones, C. *Applied Software Measurement: Assuring Productivity and Quality*. New York NY: McGraw-Hill, 1991.

Law, D. "Parallel Ada in Simulation Systems. *Defense Electronics*, Vol. 24, No. 11. November 1992, pp. 35-37.

Naval Air Warfare Center. *Next Generation Computer Resources Reference Model for Project Support Environments*, Technical Report NAWCADWAR 92023-70. 1993.

Prieto-Diaz, R. and P. Freeman. "Classifying Software for Reusability," IEEE Computer Society Press, Vol. 4, No. 1, Los Alamitos, CA: January 1987.

Rozum, J. et al. *Software Measurement Concepts for Acquisition Program Managers* (CMU/SEI-92-TR-11, ESD-TR-92-11). Pittsburgh, PA: Carnegie-Mellon University, June 1992.

San Antonio I, Panel VII. *JLC Software Workshop Final Report*. 1 February 1991.

Shlaer, S. and S. Miller. "An Object-Oriented Approach to Domain Analysis." *Software Engineering Notes*, Vol. 14, No. 5. November 1989.

U.S. Department of Commerce, National Institute of Standards and Technology. *Application Portability Profile (APP): The U.S. Government's Open System Environment Profile OSE/1 Version 1.0.* Washington, D.C.: Government Printing Office, 1991.

Weiderman, N. *Ada Adoption Handbook, Compiler Evaluation and Selection*, Version 1.0 (CMU/SEI 89-TR-13, ESD-TR-89-12). Pittsburgh, PA: Carnegie-Mellon University, 1989.

LIST OF ACRONYMS AND ABBREVIATIONS

AAS	Advanced Automation System
ABET	Ada-Based Environment for Test
ACEC	Ada Compiler Evaluation Capability
ACES	Ada Compiler Evaluation System
ACM	Association for Computing Machinery
ACSE	Association Control Service Element
ACUE	Aircraft Control Unit Emulator
ACVC	Ada Compiler Validation Capability
AdaIC	Ada Information Clearinghouse
AdaJUG	Ada Joint (Services) Users Group
Ada PSE	Ada Programming Support Environment
ADP	Automatic Data Processing
AES	Ada Evaluation System
AFATDS	Advanced Field Artillery Tactical Data System
AFB	Air Force Base
AFDSRS	Air Force Defense Software Repository System
AFSC	Air Force Systems Command
AFSPACECOM	Air Force Space Command
AI	Artificial Intelligence
AIE	Ada Integrated Environment
AIS	Automated Information System
AIU	Acoustic Interface Unit
AJPO	Ada Joint Program Office
ALS	Ada Language System
ALS/N	Ada Language System/Navy
AMMWS	Advanced Millimeter Wave Seeker
AMPS	Advanced Message Processing System
ANSI	American National Standards Institute
AP	Acquisition Plan
AP	Arithmetic Processor
APB	Acquisition Program Baseline
API	Application Programming Interface
APID	Application Programming Instructional Department
APP	Application Portability Profile
APT	Advanced Programming Technique
ARB	Acquisition Review Board
ARLB	Ada Reuse Library Browser
ARPA	Advanced Research Projects Agency
ASEET	Ada Software Engineering Education and Training

ASI	Application Software Interface
ASIS	Ada Semantic Interface Specification
ASP	Acquisition Strategy Plan
ASR	Ada Software Repository
ASSET	Asset Source for Software Engineering Technology
AST	Advanced Systems Technology
ASW	Anti-Submarine Warfare
ASWSOW	Anti-Submarine Warfare Standoff Weapon
AT&T	American Telephone & Telegraph
ATCCS	Army Tactical Command and Control System
ATD	Aircrew Training Device
ATE	Automated Test Equipment
ATF	Advanced Tactical Fighter
ATIP	Ada Technology Insertion Program
ATIS	A Tool Integration Standard
ATRIM	Aviation Training and Readiness System
AVF	Ada Validation Facility
BAFO	Best and Final Offer
BBS	Bulletin Board System
BMS	Broadcast Message Server
BP	Backplane
C2	Command and Control
C2I	Command, Control, and Intelligence
C3I	Command, Control, Communications, and Intelligence
C4I	Command, Control, Communications, Computers, and Intelligence
CAB	Common Ada Baseline
CAD	Computer-Aided Design
CAI	Computer-Aided Instruction
CAIS	Common Ada PSE Interface Set
CALS	Computer-aided Acquisition and Logistics Support
CAM	Computer-Aided Manufacture
CAMP	Common Ada Missile Packages
CARDS	Central Archive for Reusable Defense Software Program
CASE	Computer-Aided Software Engineering
CAS REPS	Casualty Reporting System
CAUWG	Commercial Ada Users Working Group
CAXI	Common Ada XWindow Interface
CC&I	Command, Control, and Intelligence
CCITT	International Consultative Committee for Telegraph and

CCITT	International Consultative Committee for Telegraph and Telephone
CCP	Code Counting Program
CCS	Combat Control System
CDA	Central Design Activity
CDB	Central Data Base
CDIF	CASE Data Interchange Format
CDPA	Central Design Programming Activity
CDR	Critical Design Review
CDRL	Contract Data Requirements List
CECOM	Communications Electronics Command
CERT	Computer Emergency Response Team
CERT/CC	Computer Emergency Response Team Coordination Center
CFE	Contractor-Furnished Equipment
CGI	Computer Graphics Interface
CGM	Computer Graphics Metafile
CI	Configuration Item
CIF	Central Issue Facility
CIM	Corporate Information Management
CLNP	Connectionless Network Protocol
CLOC	Compiled/Assembled Lines of Code
CMM	Capability Maturity Model
CMP	CoMPleteness
CMS-2	Compiler Monitor System-2
CMU	Carnegie-Mellon University
CMU/SEI	Carnegie-Mellon University/Software Engineering Institute
CNO	Chief of Naval Operations
COBOL	Common Business Oriented Language
COE	Common Operating Environment
COEA	Cost and Operational Effectiveness Analysis
COMNAVCOMTELCOM	Commander, Naval Computer and Telecommunications Command
COMSPAWARSYSCOM	Commander, Space and Naval Warfare Systems Command
CONOPS	Concept of Operations
CORBA	Common Object Request Broker Architecture
COTS	Commercial Off-The-Shelf
CPDL	Computer Program Development Laboratory
CPP	Command Program Processor
CPU	Central Processing Unit
CRADA	Cooperative Research and Development Agreement
CREASE	Catalog of Resources for Education in Ada and Software Engineering

CRISD	Computer Resource Integrated Software Document
CRLCMP	Computer Resources Life-Cycle Management Plan
CRSS	C3I Reusable Software System
CRWG	Computer Resources Working Group
CSC	Computer Sciences Corporation
CSCI	Computer Software Configuration Item
CSRO	Center for Software Reuse Operations
CSS	Centralized Structure Store
CSU	Computer Software Unit
CWG	Coordinator Working Group
D&V	Demonstration & Validation
DAB	Defense Acquisition Board
DACS	Data and Analysis Center for Software
DARPA	Defense Advanced Research Projects Agency
DAT	Digital Audio Tape
DBMS	Database Management System
DC	Device Coordinate
DCDS	Distributed Computing Design System
DCE	Distributed Computing Environment
DDI	Directorate of Defense Information
DDN	Defense Data Network
DDR&E	Director of Defense Research and Engineering
DDRS	DOD Data Repository System
DE1	Data Elements in the Source
DEM	Digitized Electronic Module
DEMVAL	Demonstration and Validation
DFCS	Digital Flight Control System
DFU	De Facto Usage
DID	Data Item Description
DISA	Defense Information Systems Agency
DMRD	Defense Management Review Decision
DOD	Department of Defense
DODD	Department of Defense Directive
DODI	Department of Defense Initiative
DON	Department of the Navy
DPI	Data Processing Installation
DP/DGU	Distributed Processor/Display Generator Unit
DRPM	Direct Reporting Program Manager
DS	Directory Service
DSRS	Defense Software Repository System
DTC 2	Desk Top Computer 2

DTN	Data Transfer Network
DTIC	Defense Technical Information Center
DUS	Design Unit Specification
DWS	Defensive Weapon System
ECCM	Electronic Counter-Countermeasures
ECLD	Embedded Comment Lines in Data
ECLS	Embedded Comment Lines in Source
ECM	Electronic Countermeasures
ECMA	European Computer Manufacturing Association
ECS	Electronic Customer Services
EDI	Electronic Data Interchange
EDL	Event-Driven Language
EDSI	Equivalent Delivered Source Instructions
EMPM	Electronic Manuscript Preparation and Markup
EMR	Extended Memory Reach
ENB	Engineering Notebook
EPROM	Erasable Programmable Read Only Memory
EP	Enhanced Processor
ERA	Entity Relationship Attribute
ESD	Electronic Systems Division
ESM	Electronic Support Measure
4GL	Fourth Generation Language
FAA	Federal Aviation Administration
FAR	Federal Acquisition Regulations
FAU	Fin Actuator Unit
FCDSSA	Fleet Combat Direction System Support Activity
FE	Functional Element
FFP	Firm Fixed Price
FFRDC	Federally Funded Research and Development Center
FIFO	First In First Out
FIPS	Federal Information Processing Standards
FIT	Flight Instrument Trainer
FMSO	Fleet Material Support Office
FP	Function Point
FPI	Functional Process Improvement
FRAWG	Front Range Ada Working Group
FSD	Full-Scale Development
FTAM	File Transfer, Access, and Management

FTP	File Transfer Program
ftp	File Transfer Protocol
43RSS	AN/UYK-43(V) Run-Time Support System
GAO	General Accounting Office
GB	Gigabyte
GEU	Guidance Electronics Unit
GFE	Government-Furnished Equipment
GFS	Government-Furnished Software
GIS	Geographic Information System
GKS	Graphical Kernel System
GM	Global Memory
GNCP	Guidance, Navigation, and Control Program
GNMP	Government Network Management Profile
GOSIP	Government Open Systems Interconnection Profile
GOTS	Government-Off-the-Shelf
GPEF	Generic Package of Elementary Functions
GPPF	Generic Package of Primitive Functions
GPO	Government Printing Office
GRACE™	Generic Reusable Ada Components for Engineering
GSIS	Graphics System Interface Standard
GTRIMS	Ground Controller Training System
GUI	Graphical User Interface
HOL	High Order Language
HP	Hewlett-Packard
HP VUE	Hewlett-Packard Visual User Environment
HPBP	High-Performance Backplane
HPP	High-Performance Processor
IBM	International Business Machines
I-CASE	Integrated Computer-Aided Software Engineering
ICC	Irvine Compiler Corporation
ICE	Independent Cost Estimate
IDEF	Integrated System Definition Language
IEC	International Electro-Technical Committee
IEEE	Institute of Electrical and Electronics Engineers
IGES	Initial Graphics Exchange Specification
IGRV	Improved Guard Rail Five
ILSP	Integrated Logistics Support Plan
IMU	Inertial Measurement Unit
INEL	Idaho National Engineering Laboratory

INFOSEC	Information System Security
InProc	In Processing
I/O	Input/Output
IOC	Initial Operating Capability
IOP	Input/Output Processor
IPO	Information Planning and Organizing
IPR	In-Process Review
IPS	Integrated Project Summary
IPSE	Integrated Project Support Environment
IRAC	International Requirements and Design Criteria
IRDS	Information Resource Dictionary System
IRS	Interface Requirements Specification
ISA	Instruction Set Architecture
ISC	Input Signal Conditioner
ISDN	Integrated Services Digital Network
ISEA	In-Service Engineering Activity
ISEE	Integrated Software Engineering Environment
ISO	International Organization for Standardization
ISSC	Information System Software Center
ITPB	Information Technology Policy Board
ITS	Integrated Test Software
IV&V	Independent Verification and Validation
JCS	Joint Chiefs of Staff
JIAWG	Joint Integrated Avionics Working Group
JIEO	Joint Interoperability and Engineering Organization
JLC-JPCG-CRM	Joint Logistics Commanders Joint Policy Coordinating Group on Computer Resources Management
JTC	Joint Technical Committee
K	1,000
KAPSE	Kernel Ada Programming Support Environment
LAN	Local Area Network
LCM	Life-Cycle Management
LCSA	Life-Cycle Support Activity
LOC	Level of Consensus
LRFP	Logistics Requirements Funding Plan
MAPSE	Minimal Ada Programming Support Environment
MAT	MATurity
MB	Megabyte

MCCDC	Marine Corps Combat Development Command
MCCR	Mission-Critical Computer Resources
MCCRES	Marine Corps Combat Readiness Evaluation System
MCO	Marine Corps Order
MENS	Mission Element Need Statement
MEPS	Message Edit Processing System
MHS	Message Handling Service
MIL-HDBK	Military Handbook
MIL-STD	Military Standard
MIMMS	Marine Corps Integrated Maintenance Management System
MIPS	Millions of Instructions per Second
MIS	Management Information System
mm	Millimeter
MMI	Man-Machine Interface
MMS	Minimum Mode Software
MOA	Memorandum of Agreement
MOTS	Military Off-The-Shelf
MSE	Master's in Software Engineering
MT	Mission Trainer
NA	Network Adaptor
NAC	Naval Avionics Center
NADC	Naval Air Development Center
NAPI	North American Portable Common Tool Environment Initiative
NAPUG	North American PCTE User's Group
NARDAC	Navy Regional Data Automation Center
NASA	National Aeronautics and Space Administration
NASEE	NAVAIR Software Engineering Environment
NATO	North Atlantic Treaty Organization
NAUG	Navy Ada Users Group
NAVAIR	Naval Air Systems Command
NAVCOMTELCOM	Naval Computer and Telecommunications Command
NAVDAC	Navy Data Automation Command
NAVSEA	Naval Sea Systems Command
NAVSUP	Naval Supply Systems Command
NAVSWC	Naval Surface Warfare Center
NAWC-AD-WAR	Naval Air Warfare Center, Aircraft Division, Warminster

NCA	Naval Center for Cost Analysis
NCCOSC	Naval Command, Control, and Ocean Surveillance Center
NCS	Network Computing Service
NCTAMS	Naval Computer and Telecommunications Area Master Station
NCTAMS LANT	NCTAMS Atlantic
NCTAMS EASTPAC	NCTAMS Eastern Pacific
NCTC	Naval Computer and Telecommunications Command
NCTS	Naval Computer and Telecommunications Station
NDC	Normalized Device Coordinate
NDI	Nondevelopmental Item
NGCR	Next Generation Computer Resources
NISBS	NATO Interoperable Submarine Broadcast System
NIST	National Institute of Standards and Technology
NISMC	Naval Information System Management Center
NISO	National Information Standards Organization
NIUF	North American ISDN Users' Forum
NM	Network Management
NOSC	Naval Ocean Systems Center
NRaD	Naval Research and Development
NSWC	Naval Surface Weapons Center
NTCSS	Naval Tactical Combat Support System
NTIS	National Technical Information Service
NUSC	Naval Undersea Command
NWRC	Navy Wide Reuse Center
NWSUS	Navy WWMCCS Site-Unique Software
OAS	Offensive Avionics System
OASD	Office of the Assistant Secretary of Defense
OCD	Operational Concept Document
OFPS	Operational Flight Program Size
OMG	Object Management Group
OMU	Operational Mock-up
OOD	Object-Oriented Design
OOP	Object-Oriented Programming
OORA	Object-Oriented Requirements Analysis
OPE	Open Systems Environment
OPNAVINST	Naval Operations Instruction
OPR	Office of Primary Responsibility
ORG	Organization Chain of Command
OS	Operating System
OSA	Open Systems Architecture

OSD	Office of the Secretary of Defense
OSE	Open Systems Environment
OSF	Open Software Foundation
OSI	Open Systems Interconnection
OSISL	Open Systems Interface Standards List
OSS	Operations Support System
OSSWG	Operating Systems Standards Working Group
PAV	Product AVailability
PC	Personal Computer
PCIS	Portable Common Interface Set
PCTE	Portable Common Tool Environment
PDL	Program Design Language
PDR	Preliminary Design Review
PDS	Post-Deployment Support
PDSS	Post-Deployment Software Support
PDU	Pulse Driver Unit
PEO	Program Executive Office
PHIGS	Programmer's Hierarchical Interactive Graphics System
PII	Protocol Independent Interface
PIMB	PCTE Interface Management Board
PIWG	Performance Issues Working Group
PMC	Project Management Charter
POC	Point of Contact
POM	Program Objective Memorandum
POSIX	Portable Operating System Interface for Computer Systems
PPBS	Planning, Programming, and Budgeting System
PRISM	Portable Reusable Integrated Software Modules
PRL	PRoblems/Limitations
PRR	Product Readiness Review
PSE	Project (or Programming) Support Environment
PSERM	Project Support Environment Reference Model
PSESWG	Project Support Environment Standard Working Group
PSA	Program Structure Analysis
PSL	Program Structure Language
R&D	Research and Development
RACS	Registration and Access Control System
RADC	Requirements and Design Criteria
RAM	Random Access Memory
RAPID	Reusable Ada Products for Information Systems Development

RCL	RAPID Center Library
RDA	Remote Database Access
RDBMS	Relational Database Management System
RDT&E	Research, Development, Test, and Evaluation
RES	Resources
REVIC	Revised Intermediate COCOMO
RFP	Request for Proposals
RLF	Reuse Library Framework
RLT	Reuse Library Toolset
RMA	Rate Monotonic Analysis
RMC	Reconfigurable Mission Computer
ROI	Return on Investment
ROM	Read Only Memory
RPC	Remote Process Communication
RPC	Remote Procedure Call
RSC	Reusable (Ada) Software Component
RTAda	Run-Time Ada
RTE	Run-Time Environment
SAE	Software Architectures Engineering
SAFENET	Survivable Adaptable Fiber-optic Embedded Network
SAI	Software Action Item
SAIL	System Avionics Integration Laboratory
SAME	SQL Ada Module Extension
SAMeDL	SQL Ada Module Description Language
SASET	Software Architecture Sizing and Estimating Tool
SASSY	Supported Activities Supply System
SCAI	Space Command & Control Architecture Infrastructure
SCCS	Submarine Combat Control System
SCE	Software Capability Evaluation
SCH	Scheduler
SCL	Stand-alone Comment Lines
SCMP	System Configuration Management Plan
SCRB	Software Change Review Board
SCS	Submarine Combat System
SDC-W	Software Development Center, Washington
SDD	System Design Definition
SDE	Software Development Environment
SDF	Software Development Folder
SDIO	Strategic Defense Initiative Organization
SDL	Software Development Laboratory
SDP	Software Development Plan

SDP	System Division Paper
SDR	System Design Review
SDSR	Software Development Status Report
SDTS	Spatial Data Transfer Standard
SECNAVINST	Secretary of the Navy Instruction
SECNAVNOTE	Secretary of the Navy Note
SECR	Standard Embedded Computer Resource
SEE	Software Engineering Environment
SEI	Software Engineering Institute
SEM	Standard Electronic Module
SEMP	System Engineering Management Plan
SEO	Software Executive Official
SEOC	Software Executive Official Council
SEPG	Software Engineering Process Group
SES	Senior Executive Service
SGS/AC	Shipboard Gridlock System with Auto-Correlation
SGML	Standard Generalized Markup Language
SIGAda	Special Interest Group on Ada
SIGSOFT	Special Interest Group on Software Engineering
SIL	System Integration Laboratory
SIP	System Integration Plan
SISTO	Software and Intelligent Systems Technology Office
SLOC	Source Lines of Code
SLOC/SM	Source Lines of Code per Staff Month
SLOCWC	Source Lines of Code Without Comments
SMB	Submarine Message Buffer
SMM	Software Management Metrics
SMP	Software Master Plan
SOW	Statement of Work
SPA	Software Process Assessment
SPAWAR	Space and Naval Warfare Systems Command
SPC	Software Productivity Consortium
SPD	Software Process Definition
SPDL	Standard Page Description Language
SPI	Software Process Improvement
SPO	System Programming Office
SPR	Software Problem Report
SQAP	Software Quality Assurance Plan
SQL	Structured Query Language
SRC	Software Requirements Change
SRR	Software Requirements Review
SRS	Software Requirements Specification

SSA	Software Support Activity
SSC	ystem Support Center
SSS	System/Segment Specification
STANFINS	Standard Financial System
STANFINS-R	Standard Financial System Redesign
STARFIARS	Standard Army Financial Accounting and Reporting System
STARS	Software Technology for Adaptable, Reliable Systems
STB	STaBility
STC	Software Technology Conference
STEP	Standard for the Exchange of Product Model Data
STI	Software Technology Initiative
STSC	Software Technoiogy Support Center
SUP	Support Planning
SWAP	Software Action Plan
SWAP-WG	Software Action Plan Working Group
SWG	Special Working Group
SWTP	Software Technology Plan
SYSCOM	Systems Command
TAC	Tactical Advanced Computer
TACAMO	Take Charge and Move Out
TACFIRE	Tactical Fire Direction
TAFIM	Technical Architecture For Information Management
TADSTAND	Tactical Digital Standard
TAMPS	Tactical Aircraft Mission Planning System
TC	Target Capacity
TC	Technical Committee
TCL	Total Comment Lines
TCP/IP	Transmission Control Protocol/Internet Protocol
TD	Technical Directive
TDA	Technical Directive Authority
TDT	Theater Display Terminal
TEMP	Test and Evaluation Master Plan
TEP	Test and Evaluation Plan
TFA	Transparent File Access
TLCSC/LLCSC	Top Level/Lower Level Computer Software Component
TLOC	Total Lines of Code
TOES	Telephone Order-Entry System

TOPS	Training and Operations Section
TQM	Total Quality Management
TSGCEE	Tri-Service Group on Communications and Electronics Equipment
UIMS	User Interface Management System
ULLS	Unit Level Logistics System
USMC	U.S. Marine Corps
USTAG	United States Technical Advisory Group
USW	Undersea Warfare
UUT	Unit Under Test
VADS	Verdix Ada Development System
VDI	Virtual Device Interface
VHSIC	Very High-Speed Integrated Circuit
VRC	Virtual Reference Coordinate
VSR	Validation Summary Report
VT	Virtual Terminal
WAdaS	Washington Ada Symposium
WAM	WWMCCS ADP Modernization
WBS	Work Breakdown Structure
WC	World Coordinate
WFNIA	Wells Fargo Nikko Investment Advisors
WIS	WWMCCS Information System
WPAFB	Wright Patterson Air Force Base
WST	Weapon System Trainer
WWMCCS	World Wide Military Command and Control System

Glossary

The definitions provided below are used throughout the Department of Defense (DOD) software community. Most of the definitions are from other sources. The source from which the definition originated is indicated in brackets at the end of the definition, and the complete information about the source is provided at the end of this glossary. The originator of these definitions may periodically modify the definition.

ACTIVITY. (1) A task or discrete step in a methodology performed to provide management, technical, and support information for decision making. [I-CASE] (2) A unit of work to be completed in achieving the objective of a software project. (3) The lowest level of the hierarchical decomposition of functions in the Enterprise Model. [I-CASE] (4) A defined portion of work within a project that typically has a designated owner, entry requirements, implementation requirements, exit requirements, duration, and schedules. Examples of activities include developing the product specification document, creating the high-level design, coding, and performing the system test.

ACCURACY. (1) A quality of that which is free of error. [ISO] (2) A qualitative assessment of freedom from error, a high assessment corresponding to a small error. [ISO] (3) A quantitative measure of the magnitude of error, preferably expressed as a function of the relative error, a high value of this measure corresponding to a small error [ISO]. (4) A quantitative assessment of freedom from error. [IEEE]

Ada. A programming language developed on behalf of the U.S. DOD for use in large, real-time, embedded computer systems.

Ada BINDINGS. Software linkages between the Ada programming language and other languages. For example, Ada-SQL bindings provide the means to link Ada programs with a Relational Database Management System (RDBMS) using SQL or SQL-like embedded instructions.

Ada PROGRAMMING SUPPORT ENVIRONMENT (AdaPSE). A set or hardware and software that supports all aspects of software development and maintenance, including project management and configuration management.

Ada REPOSITORY. A database that contains reusable software components for information system development. The Reusable Ada Products for Information Systems Development Program (RAPID) was developed by the U.S. Army and purchased by the Standard Systems Center. [I-CASE]

ADAPTABILITY. The ease with which software allows differing system constraints and user needs to be satisfied. [IEEE]

ALGORITHM. A finite set of well-defined rules for the solution of a problem in a finite number of steps (e.g., a complete specification of a sequence of arithmetic operations for evaluating sin x to a given precision). [ISO] (2) A finite set of well-defined rules that gives a sequence of operations for performing a specific task. [IEEE]

ARCHITECTURE. (1) The structure of components, their interrelationships, and the principles and guidelines governing their design and evolution over time. (2) Organization structure of a system or component. [IEEE]

ARCHITECTURAL DESIGN. (1) The process of defining a collection of hardware and software components and their interfaces to establish a framework for the development of a computer system. [IEEE] (2) The result of the architectural design process. [IEEE]

ARTIFICIAL INTELLIGENCE. A subfield within computer science concerned with developing technology to enable computers to solve problems (or assist humans in solving problems) by using explicit representations of knowledge and reasoning methods employing that knowledge. [SWTP]

ATTRIBUTE. A piece of information about an entity or relationship. [I-CASE].

AUTOMATED APPLICATION GENERATORS. Programs that generate application programs from special application-oriented languages. [SWTP]

AUTOMATED INFORMATION SYSTEM (AIS). An automated system in which the data stored will be used in spontaneous ways that are not fully predictable in advance for obtaining information.

AVAILABILITY. (1) The probability that software will be able to perform its designated system function when required for use. [IEEE] (2) The ratio of system up-time to total operating time. [IEEE] (3) The ability of an item to perform its designated function when required for use. [ANSI/ASQC]

BASELINE. (1) A configuration identification document or set of such documents (regardless of media) formally designed and fixed at a specific time during a configuration item's life cycle. Baselines, plus approved changes to those baselines, constitute the current configuration identification. [DOD-HDBK-287A] (2) A specification or product that has been formally reviewed and agreed upon, that

thereafter serves as the basis for further development and that can be changed only through formal change control procedures. [IEEE] (3) A document or set of such documents formally designated and fixed at a specific time during the life cycle of a configuration item. [IEEE]

BSY-2. The distributed battle management system for the SSN-21 submarine in the U.S. Navy. The system integrates sensors and weapons and displays information. [SWTP]

C2. *See* command and control.

C3I. *See* definitions of command and control, communications, and intelligence.

CASE. Computer-Aided Software Engineering is the automation of well-defined methodologies that are used in the development and maintenance of software products. These methodologies apply to nearly every process or activity of a product development cycle (e.g., project planning and tracking, product designing, coding, and testing).

CASE ENVIRONMENT. A CASE environment is a computer system consisting of a fixed set of core facilities that form the environment framework and a set of facilities, called tools, that supports software development activities. [I-CASE]

CERTIFICATION. (1) A written guarantee that a system or computer program complies with its specified requirements. [IEEE] (2) A written authorization that states that a computer system is secure and is permitted to operate in a defined environment with or producing sensitive information. [IEEE] (3) The formal demonstration of system acceptability to obtain authorization for its operational use. [IEEE] (4) The process of confirming that a system, software subsystem, or computer program is capable of satisfying its specified requirements in an operational environment. Certification usually occurs in the field under actual conditions and is used to evaluate not only the software itself but also the specifications to which the software was constructed. Certification extends the process of verification and validation to an actual or simulated operational environment. [IEEE] (5) The procedure and action by a duly authorized body of determining, verifying, and attesting in writing to the qualifications of personnel, processes, procedures, or items in accordance with applicable requirements. [ANSI/ASQC]

CHANGE CONTROL PROCESS. A defined process to be followed when a change to a controlled document or procedure is proposed. A typical use is to control proposed changes to product objectives or product specifications once these documents have been approved.

CLIENT-SERVER ENVIRONMENT. Defined most broadly, a client-server environment involves a requester of a service or services and a deliverer of those services. The requester is normally a user on a microcomputer, whereas the deliverer is usually a file server. Toronto-based International Data Corp. analyst Michael O'Neill defines client-server as network architectures in which applications are run to some extent on desktop computers and servers provide additional services such as storage, applications, communications, and printing. [I-CASE]

CODING. The act of writing instructions that are immediately computer recognizable or can be assembled or compiled to form computer-recognizable instructions. Within a product development cycle, this activity follows the low-level design activity and precedes the unit testing and function testing activities.

COMMAND AND CONTROL. The provision of communications and intelligence to a properly designated commander of assigned forces for use in the accomplishment of a mission. Command and control functions are performed through an arrangement of personnel, equipment, communications, facilities, and procedures employed by a commander in planning, directing, coordinating, and controlling forces and operations in the accomplishment of the mission. [PUB 1-02]

COMMERCIAL-OFF-THE-SHELF (COTS) PRODUCTS. Items regularly used in the course of normal business operations for other than Government purposes that (a) have been sold or licensed to the general public; (b) have not been sold or licensed, but have been offered for sale or license to the general public; (c) are not yet available in the commercial marketplace, but will be available for commercial delivery in a reasonable period of time; (d) are as described in (a), (b), or (c) that would require only minor modification in order to meet the requirements of the procuring agency. Minor modification means a modification to a commercial item that does not alter the commercial item's function or essential physical characteristics. [I-CASE]

COMMON INFORMATION REPOSITORY. A database that contains all of the information pertaining to systems development and provides the means for all tools in the development environment to share information engineering information. [I-CASE]

COMMUNICATIONS. A method or means of conveying information of any kind from one person or place to another [Webster].

COMPLEXITY METRICS. Pertaining to any of a set of structure-based metrics that measures the degree to which a system or component has a design or implementation that is difficult to understand and verify (See SoftwareComplexity). The more

complex a program, the more likely it will contain errors. Examples of complexity metrics are module size, Halstead's Software Science Metrics, McCabe's cyclomatic number, and McClure's control variable complexity. [I-CASE]

COMPONENT. A portion of a software system that contains one logical subdivision of the system. A given software system may be seen as composed of its various components. [SWTP]

COMPUTER SOFTWARE CONFIGURATION ITEM (CSCI). A configuration item for computer software. [DOD-STD-2167A]

CONCEPT OF OPERATIONS (CONOPS). Detailed narrative description of a targeted area's functions and all of their interrelationships during peacetime, exercise, crisis, and wartime.

CONCEPTUAL DATA MODEL. Identification and description of all of the entities and relationships that support the key areas, tasks, and activities defined in the Enterprise Model of the targeted area. These entities and relationships define the inherent structure of information within the enterprise. [I-CASE]

CONFIGURATION CONTROL. The systematic evaluation, approval or disapproval, and implementation of all approved changes in the configuration of a configuration item after formal establishment of its configuration identification. [I-CASE]

CONFIGURATION ITEM. Hardware or software, or an aggregate of both, which is designated by the contracting agency (or project configuration manager) for configuration management. [DOD-HDBK-287A]

CONFIGURATION IDENTIFICATION. The approved or conditionally approved technical documentation for a configuration item as set forth in specifications, drawings, associated lists, and documents referenced therein. [DOD-HDBK-287A]

CONFIGURATION MANAGEMENT. (1) The process of identifying and defining the configuration items in a system, controlling the release and change of these items throughout the system life cycle, recording and reporting the status of configuration items and change requests, and verifying the completeness and correctness of configuration items. [IEEE] (2) A discipline applying technical and administrative direction and surveillance to (a) identify and document the functional and physical characteristics of a configuration item, (b) control changes to those characteristics, and (c) record and report change processing and implementation status. [DOD-STD 480A]

CONFIGURATION MANAGEMENT PLAN (CMP). The Configuration Management Plan defines the implementation (including policies and methods) of configuration management on a particular program/project. [DOD-HDBK-287A]

CONTRACT CHANGE PROPOSAL. A formal priced document also referred to as "Task Change Proposal (TCP)" used to propose changes to the scope of work of the contract. It is different from an Engineering Change Proposal (ECP) in that it does not affect specifications or drawing requirements. It may be used to propose changes to contractual plans, the Statement of Work (SOW), Contract Data Requirements List (CDRL), and others. [I-CASE]

CONTRACT DATA REQUIREMENTS LIST (CDRL). Identifies the name of the item to be delivered, the date it is to be delivered, and a Data Item Description (DID).

CONTRACTOR. An individual, partnership, company, corporation, or association that has a contract with the contracting agency (Government) for the design, development, maintenance, modification, or supply of configuration items and services under the terms of a contract. A Government agency performing any of the above actions is considered a "contractor" for configuration management purposes. [DOD-HDBK-287A]

COST ANALYSIS. (1) The act of breaking down a cost summary into its constituents and studying and reporting on each factor. (2) The comparison of costs (as of a standard with actual or for a given period with another) for the purpose of disclosing and reporting on conditions subject to improvement. [WEBSTER]

CRITICAL PATH. The collection of work activities in a product development cycle that are neck-to-neck with one another and define the longest duration for a project.

CRITICAL REAL-TIME SYSTEM. A system in which failure to meet a deadline has an impact on safety or could cause large financial, social, or military losses. [SWTP]

CRITICAL REQUIREMENTS. Any or all of a wide range of characteristics the absence or diminished presence of which in a system can result in serious consequences. [SWTP]

DATA. A representation of facts, concepts, or instructions in a formalized manner suitable for communication, interpretation, or processing by human or automatic means. Any representations such as characters or analog quantities to which meaning is or might be assigned. [PUB 1-02]

DATABASES. Electronic repositories of information accessible through a query language interface. [SWTP]

DATABASE MANAGEMENT SYSTEM (DBMS). An integrated software system that has facilities for defining the logical and physical structure of data in a database and for accessing, entering, and deleting data. [DOC]

DATA FLOW DIAGRAM. A graphical representation of the various processes and flows of information that make up a function. [I-CASE]

DATA INTEGRITY. Confidence that the data a software system is using or producing is right, data values being produced are right, and spurious values are not being inserted into the system from external sources. [SWTP]

DATA ITEM DESCRIPTION (DID). A document that contains details and reference standards with which a deliverable must comply. [I-CASE]

DATA MODELING. The process of identifying an application's data elements, data structures, and file format structures. This includes delineating the relationships between data elements, generally with entity-relationship diagrams. [I-CASE]

DATA REPOSITORIES. Refers to the dynamic data that are used during the development process. Repositories will contain a broad range of fine and coarse grain project information. [I-CASE]

DESIGN. The process of defining the architecture, components, interfaces, and other characteristics of a system or component; often referred to as the Design Phase. [I-CASE]

DESIGN FAULT. A fault committed during the design of a software system or during a subsequent modification of the design. [SWTP]

DEVELOPMENT ENVIRONMENT. The hardware and software platforms that will be used to support software development activities. [I-CASE]

DISTRIBUTED PROCESSING. The organization of processing to be carried out on a distributed system (see distributed system). Each process is free to process local data and make local decisions. The processes exchange information with each other over a data communication network to process data or to read decisions that affect multiple processes. [DOC] Running a program on several different machines. [SWTP]

DISTRIBUTED SYSTEM. Any system in which a number of independent interconnected computers can cooperate. [DOC]

DOCUMENT. Any subset of technical data that, packaged for delivery on a single medium, meets the format, content, consistency, and completeness requirements of a unified control specification whether delivered in hard copy or digital form. A document is a self-contained body of engineering data that supports, alone or with other documents, an engineering or maintenance function. [I-CASE]

DOMAIN. A set of current and future applications that performs common sets of functions. DOD software domains include avionics, vehicle control, C3I, and logistics. [SWTP]

DOMAIN ANALYSIS. The process of identifying, collecting, organizing, and representing the relevant information in a domain. The process is based on the study of existing systems and their development histories, knowledge captured from domain experts, underlying theory, and emerging technology within the domain. [SWTP]

DOMAIN KNOWLEDGE. (1) Knowledge about the objects, concepts, and relationships of a particular domain. [SWTP] (2) The set of data and business rules applicable to a product application or functional area. [I-CASE]

DOMAIN-SPECIFIC. Concerned with the objects, concepts, and relationships of a particular domain. [SWTP]

EMBEDDED COMPUTER SYSTEM. A computer system that is integral to a larger system the primary purpose of which is not computational, (e.g., a computer system in a weapon, aircraft, command and control, or rapid transit system). [IEEE]

EMBEDDED SOFTWARE. Software for an embedded computer system. [IEEE]

ENTERPRISE MODEL. (1) A hierarchical description of key areas of functionality, their subordinate tasks, and specific activities identified in the Concept of Operations (CONOPS). (2) A high-level model of an organization's information architecture that consists of a function model and a data model. [ELG] (3) A description of the (entity) types, functions, and processes. [MARTIN]

ENTITY. (1) A distinct thing (such as a person, place, thing, or event) of interest to the enterprise about which data are stored. It holds meaning in context with other entities in the enterprise. (2) Person, place, thing, concept, event, or activity about which an organization wishes to keep information. [DODM]

ENVIRONMENT. (1) Collection of hardware and software enveloped in a process to engineer software. (2) Everything that supports a system or the performance of a function. [ELG] (3) The conditions that affect the performance of a system or function. [ELG] (4) That part of the real world that contains the users that exchange messages with an information system. [ELG]

ENVIRONMENT FRAMEWORK. The core set of facilities in a CASE environment that provides necessary control, data, presentation, and communication services for tools executing in that environment. [I-CASE]

ERROR. (1) A manifestation of a fault; data (usually involving the system state) that produce a failure when processed by the system. (2) A discrepancy between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition. [IEEE]

EVOLUTIONARY PROTOTYPING. An approach to software development where a user's system is established using rapid prototyping techniques, then is updated and maintained as required. [I-CASE]

EXCEPTION. An event that causes suspension of normal program execution. [IEEE]

EXPERT SYSTEMS. Computer programs built for commercial application by using the programming techniques of Artificial Intelligence (AI), especially those techniques developed for problem solving, and involving the use of appropriate information acquired previously from human experts. [DOC]

FAILURE. (1) The result of a system or component not performing a required function within specified constraints. (2) The termination of the ability of a functional unit to perform its required function. (3) The inability of a system or system component to perform a required function within specified limits (may be produced when a fault is encountered). (4) A departure of program operation from program requirements. [IEEE]

FAILURE RATE. (1) The ratio of the number of failures to a given unit of measure (e.g., failures per unit of time, failures per number of transactions, failures per number of computer runs). (2) In reliability modeling, the ratio of the number of failures of a given category or severity to a given period of time (e.g., failures per second of execution time, failures per month). [IEEE]

FAULT. (1) An accidental condition that causes a functional unit to fail to perform its required function. (2) A manifestation of an error in software. A fault,

if encountered, may cause a failure. [IEEE] (3) A discrepancy in an automated system encountered during testing. [I-CASE]

FAULT AVOIDANCE. (1) Action that reduces the likelihood of a fault during system operation, such as isolating or replacing system components with a high predicted probability of failure based on error reports or stress data. [SWTP] (2) The elimination of faults by careful and conservative design and construction practices. [SWTP]

FAULT PREVENTION. The elimination of faults by careful and conservative design and construction practices. [SWTP]

FAULT RECOVERY. The attempt to bring a system to a state acceptable for continued operation. [SWTP]

FAULT TOLERANCE. The built-in capacity of a system to provide continued correct execution despite faults or failures of hardware or software components. [IEEE]

FAULT-TOLERANT SOFTWARE. Software that includes functions for detecting, identifying, confining, and/or recovering from faults to create a fault-tolerant system. [SWTP]

FORMAL QUALIFICATION TEST. A test conducted in accordance with formally approved test plans and procedures and witnessed by Government representatives to show that the integrated hardware and software satisfy specific requirements. [I-CASE]

FORWARD ENGINEERING. (1) The process of generating the design from the requirements and generating the code from the design. (2) The traditional process of moving from high-level abstractions and logical, implementation-independent designs to the physical implementation of a system. [CHIK]

FRAMEWORKS. General designs or architectures for systems that can be customized or specialized for particular applications. [SWTP]

FUNCTION. An action that a product is capable of performing. For example, functions for a word processor might be define margins, define page length, set tabs, change font for all section headings, search for words and phrases, and automatic document save.

FUNCTIONAL REQUIREMENT. A requirement that specifies a function that a system or system component must be capable of performing. [IEEE]

FUNCTION TEST. The testing of each product function across one or more modules. Some amount of scaffolding is typically required to perform this test. [I-CASE]

GOVERNMENT-FURNISHED EQUIPMENT (GFE). Equipment furnished by the U.S. Government. [I-CASE]

GOVERNMENT OPEN SYSTEMS INTERCONNECTION PROFILE (GOSIP). A Federal Information Processing Standards Publication (FIPS PUB) that defines a common set of data communication protocols that enable systems developed by different vendors to interoperate and enable users of different applications on these systems to exchange information. [I-CASE]

GOSIP-COMPLIANT. Computer network protocols that are in compliance with FIPS PUB 146, "Government Open Systems Interconnection Profile." [I-CASE]

GRACEFUL DEGRADATION. The ability of a system to shed noncritical functionality to accomplish critical functions within their deadlines during failures. [SWTP]

HARD REAL-TIME SYSTEM. A system that must completely service each task and produce results within specified time intervals (i.e., a system that must respond to hard deadlines). [SWTP]

HARDWARE. Physical equipment used in data processing, as opposed to computer programs, procedures, rules, and associated documentation. [IEEE]

HETEROGENEOUS NETWORK. A network of different host computers, such as those of different manufacturers. [I-CASE]

HETEROGENEOUS PROCESSORS. Processors of different types that use different organization (e.g., shared memory, arrays) or architecture (Reduced Instruction Set Computer [RISC], Complete Instruction Set Computer [CISC]). Includes also processors built by different manufacturers (e.g., Digital Equipment Corporation, Sun, IBM). [SWTP]

HIGH ASSURANCE SOFTWARE. Software for which there is compelling evidence that the computer system will respond properly under all circumstances in the context of its application. [SWTP]

HIGH-LEVEL DESIGN. The level of design required to understand how the components of a product will technically work with one another and with the surrounding hardware and software environment in which the components must operate. This design identifies the components that make up the product, defines the functional mission for each component, and defines the interface across these components and externally to the operating environment.

HYPERTEXT. A method of organizing text to enable the user to browse through the text in any order and examine any portion at any time without having to read the text from start to end. [SWTP]

I-CASE. Integrated Computer Aided Software Engineering components that span the full software development life cycle. [I-CASE]

I-CASE ENVIRONMENT. Automated software development environment that includes a Software Engineering Environment (SEE), an Operational Test Environment (OTE), and an Application Execution Environment (AEE). It will support and be used by DOD organizations responsible for development, modernization, and maintenance of AISs. [I-CASE]

IEEE FUTUREBUS+. A computer backplane standard adopted by the U.S. Navy for its next-generation computing systems. [SWTP]

IMPLEMENTATION PHASE. That portion of a system's life cycle when the system reaches its initial operational capability. [I-CASE]

INDEPENDENT VERIFICATION AND VALIDATION (IV&V). (1) Verification and validation of a software product by an organization that is both technically and managerially separate from the organization responsible for developing the product. [IEEE] (2) Verification and validation of a software product by individuals or groups other than those who performed the original design but who may be from the same organization. The degree of independence must be a function of the importance of the software. [IEEE] (3) An independent review of the software product for functional effectiveness and technical sufficiency.

INFORMAL TEST. The testing performed on a product that is typically conducted in a loosely controlled environment and is performed by the programmers who developed the code to be tested. Both the unit test and function test are considered informal test activities. Some amount of scaffolding is typically required during the informal test period.

INFORMATION ENGINEERING. The science of analyzing value-added information usage and organizing heterogeneous information (e.g., hard copy, ASCII, graphics, voice, video, structured files) so that it is stored, processed, and retrieved in a form useful to each level of decision making within an organization. [SWTP]

INFORMATION HIDING. A design concept put forward by Parnas [Parnas 1971, 1972]. Modules most likely to be changed in the future should be designed in such a way that those design decisions are hidden from other modules. Therefore, if a change has to be made in the future, such a change is localized to one specific module. [I-CASE]

INFORMATION REPOSITORY. Encompasses the capabilities currently found in multiple types of products (i.e., data repository, repository manager, library manager, and reuse library). [I-CASE]

INFORMATION RESOURCE DICTIONARY SYSTEM (IRDS). A computer software system that provides facilities to control, describe, protect, document, and facilitate use of an installation's information resources. [I-CASE]

INFORMATION SYSTEMS ENGINEERING. The process of defining requirements for databases, applications, and technical services. [I-CASE]

INSPECTION: Examination of an activity by a group of people, typically peers, to identify and remove defects and problems from a product.

INTEGRATED LOGISTICS SUPPORT (ILS). A composite of the elements necessary to assure the effective and economical support of a system or equipment at all levels of maintenance for its programmed life cycle. [I-CASE]

INTEGRATION. The process of combining software elements, hardware elements, or both into an overall system. [IEEE]

INTEGRATION SERVICES. Services that provide for integration of components across the life cycle of software applications, from concept development through decommissioning or replacement. [I-CASE]

INTERFACE. The supporting hardware and software through which dialogue takes place. [SWTP]

KNOWLEDGE. (1) Computer data structures designed to capture or encode knowledge in structural forms that can be easily manipulated by reasoning processes. [SWTP] (2) What a person or computer must know to perform a given task. [SWTP]

KNOWLEDGE REPRESENTATION. Computer data structures designed to capture or encode knowledge in structural forms that can be easily manipulated by reasoning processes. [SWTP]

LIBRARY MANAGER. Performs version control and configuration management on larger grain objects such as source code, object modules, and documentation. [I-CASE]

LIFE CYCLE. (1) The period of time encompassing the life of an automated system. (2) The period of time that begins when a system is conceived and ends when the product is no longer available for use. [IEEE]

LIFE-CYCLE MODEL. A model of the software life cycle describing the series of steps or phases through which the software progresses (e.g., Code and Fix, Waterfall, Rapid Prototyping, and Incremental Release). [I-CASE]

LOW-LEVEL DESIGN. A term representing two levels of design: (1) The design required to understand how the modules within each component will technically work with one another. This design identifies the modules that make up each component, the functional mission of each module, and the interface across these modules. (2) The design required to define the design within each of the many modules that may constitute each component. This design level identifies each programming decision path within each module and is the lowest level of design before coding.

MAINTENANCE SUPPORT. Modification of a product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a changed environment. [IEEE]

MEASUREMENT. The act or process of measuring something to ascertain the quantity, mass, extent, or degree in terms of a standard unit or fixed amount, usually by means of an instrument or container marked off in the units. [WEBSTER]

MEGAPROGRAMMING. (1) Computer systems design and implementation using preexisting software modules of known functionality as primitives [SWTP]. (2) The development of software by composing components rather than individual lines of code. [SWTP]

METHODOLOGY. (1) The way the group activities within a process are accomplished; the approach to solving the problem; a body of methods, rules, and postulates employed by a discipline. (2) A general philosophy for carrying out a process; composed of procedures, principles, and practices. [STARS]

METRICS. (1) Numerical data collected during the software production process and used to compute measures of quality, productivity, and performance of software products, development tools, methods, techniques, and processes. (2) A composite of one or more measures used to understand a process. [BOEING] (3) Those measurements established for each step in the software engineering process that are used to determine its effectiveness. The metrics define the results of each process stage and relate them to resources expended, errors introduced, errors removed, and various coverage, efficiency, and productivity indicators. [I-CASE]

MODERNIZATION. The implementation of new techniques or technology. Through modernization, the Government seeks to redesign all antiquated software systems into Ada applications and implement them as such. [I-CASE]

MODIFICATION. The process of changing existing software to support the evolving changes in design specifications. [DOD-HDBK-287A]

MODULE. (1) Software components required to support the products and applications within the operational constraints of the environment. (2) Code that represents part of a function, a single function, or more than one function. A module is code that can be independently compiled. One or more modules usually make up a component. [I-CASE]

NETWORK SERVICES. Services to support distributed applications requiring data access and applications interoperability in heterogeneous or homogeneous networked environments. [I-CASE]

OBJECT. An entity that is characterized by the operations that can be applied on or by the entity. [I-CASE]

OBJECT MANAGEMENT SERVICES. Services that manage an information repository that stores information supporting software development of program or project data across all components within the environment. [I-CASE]

OBJECT MANAGEMENT SYSTEM. A data management system the data of which are characterized by object descriptions. [SWTP]

OBJECT-ORIENTED DESIGN. A method of design encompassing the process of object-oriented decomposition and a notation for depicting both logical and physical as well as static and dynamic models of the system under design; specifically, this notation includes class diagrams, object diagrams, module diagrams, and process diagrams.

OPEN SYSTEMS. Refers to a combination of standards. Specifically, open systems is a combination of the standards as defined in the National Institute of Standards and Technology (NIST) Application Portability Profile (APP) for POSIX and XPG3. This is applicable to both hardware and software. [I-CASE]

OPEN SYSTEMS ARCHITECTURE (OSA). An architecture that implements international standard protocols. [I-CASE]

OPEN SYSTEMS ENVIRONMENT (OSE). Computing, communications, and application software products that have vendor-independent interfaces. [ELG]

OPERATIONAL TEST ENVIRONMENT (OTE). The primary purpose of the I-CASE OTE is to test applications generated by the I-CASE SEE. This test environment is used to test application functionality and evaluate performance of the application in the I-CASE SEE (target environment). The I-CASE OTE supports compilation and testing of application source code developed and maintained in the SEE and includes run-time components required for the applications to execute on their open system hardware platforms. [I-CASE]

PHASE. A defined portion of a product development cycle. The portions defined as phases are arbitrary and are usually determined by the group that plans the project. Also, a subset of activities that make up a major activity. For example, a project document review cycle is composed of five phases: preparation, review, update, approval, and information.

PHYSICAL DATA MODEL. Representation of the technically independent data requirements in a physical environment in hardware, software, and network configurations as alternative methods of implementing the conceptual design in a "functional" system or representing them in the constraints of an existing physical environment. [I-CASE]

PORTABILITY. The ease with which software can be transferred from one computer system or environment to another. [IEEE]

PORTABLE OPERATING SYSTEM INTERFACE (POSIX). An IEEE standard that defines a C language source interface to an operating system environment. [I-CASE]

PROCESS. (1) A set of actions, tasks, and procedures that, when performed or executed, obtains a specific goal or objective. (2) A series of actions, changes or functions that achieve an end or result. (3) A model for accomplishing an objective. [BOEING] (4) The logical organization of people, machines, tools, methods, and procedures into work activities designed to produce a specified end result (work product). The term software process refers to processes that are intrinsic to developing and evolving software systems. [SWTP] (5) The grouping of activities; what is to be done in the development of software, the sequence of phases, tasks, and activities required. (6) The manner in which a software development project, or any of its many integral parts, is planned, developed, or tracked. For example, the method of logging in a problem and tracking that problem to a satisfactory closure is defined as a process. [I-CASE]

PROCESS DEFINITION. An explanation of the meaning of a specific software process (e.g., the software quality assurance process). [SWTP]

PROCESS MANAGEMENT. The management of the creation and maintenance of software process definitions before and during their use in building or maintaining systems or families of systems. [SWTP]

PROCESS MANAGEMENT SERVICES. Services that provide the mechanism to manage life-cycle processes as described by the I-CASE Software Process Model. [I-CASE]

PROCESS MODEL. One specific embodiment of a software process architecture (i.e., an embodiment of a set of software process definitions configured to create or maintain a system or family of systems). [SWTP]

PROCESS MODELING. To make or conform to a chosen framework for identifying, defining, and organizing the business strategies, rules, and processes needed to manage and support the way an organization does or wants to do business. [I-CASE]

PRODUCT. A software package, consisting of code and documentation, that is eventually delivered to a customer. In a more global sense, the definition of product also includes the product support materials related to activities such as marketing and maintenance.

PRODUCT BASELINE. The initially or conditionally approved product configuration identification. [I-CASE]

PRODUCT DEVELOPMENT CYCLE. The sequence of activities followed in developing a product. A product development cycle covers a wide range of activities

that typically include creating the product objectives and the product specifications and designing, coding, testing and packaging the final product for delivery to customers.

PRODUCT SPECIFICATIONS. A document that details precisely what the user will receive and use when the completed product is made available. Every function, command, screen, prompt, and so on is documented in the specification so that all participants involved in the product development cycle know the product they are to build, test, document, and support.

PROGRAM. The code portion of a product or test case or a collection of components linked together.

PROGRAM OR PROJECT MANAGEMENT. Provides the necessary planning, organization, staffing, direction, and control for the orderly development or acquisition of a software product. [I-CASE]

PROGRAMMING LANGUAGE. An artificial language designed to generate or express programs. [IEEE]

PROJECT. The combined resources (i.e., people, computers, and materials), processes, and activities dedicated to building and delivering a product. Also, a group of people, typically from two or more organizations, that is working on the same product.

PROTOTYPE. (1) An experimental model for a system on which decisions for later versions are based or judged. [SWTP] (2) A functional model of a target system, suitable for evaluation of design, performance, and product potential or an instance of a software version that does not exhibit all of the properties of the final system; usually lacking in terms of functional or performance attributes. [DOD-HDBK-287A] (3) A preliminary type, form or instance of a system that serves as a model for later stages or for the final, complete version of the system. [IEEE] (4) An early running model of a product the primary purpose of which is usually to experiment with, demonstrate, or prove the feasibility of a concept.

PROOF-OF-CONCEPT. A type of demonstration generating evidence that the concept is effective. [SWTP]

PROVABLY SECURE SOFTWARE SYSTEM. A software system for which it can be proved that the software has a level of assurance that the system can enforce a specific security policy and that integrity, availability, and liveness are also assured. [SWTP]

QUALIFICATION TESTING. Product testing that demonstrates that the products satisfies all "MINIMUM" requirements in the current version of the I-CASE Specification Requirements and that occurs, at the Government's option, as a precondition for the product baseline. [I-CASE]

QUALITY. Conformance to requirements. Once the product and process requirements have been defined, the quality can be measured for compliance.

QUALITY ASSURANCE (QA). (1) A planned and systematic pattern of all actions necessary to provide adequate confidence that the item or product conforms to established technical requirements. [IEEE] (2) Methodologies and techniques used to perform critical evaluation of computer systems to ensure release of high-quality products. [I-CASE]

QUALITY ASSURANCE GROUP. People assigned to perform an "outside check-and-balance" role of ensuring that a product is being developed according to an acceptable process.

QUALITY ASSURANCE PLAN. A document that can be used to define, track, and measure both product and process quality goals throughout the product development cycle.

RAPID PROTOTYPING. (1) The process of building executable versions of partially constructed systems to allow early observation and understanding of system behavior, especially its interface. [SWTP] (2) The use of an application generator and/or nonprocedural Fourth Generation Languages (4GLs) to quickly develop a prototype of key portions of the user's desired capability. [I-CASE]

REAL-TIME SOFTWARE SYSTEM. A software system that satisfies critical timing requirements. The correctness of the software depends on the results of computation and on the time at which the results are produced. [SWTP]

REENGINEERING. (1) The examination and alteration of a subject system to reconstitute it in a new form and the subsequent implementation of the new form; generally includes some form of reverse engineering (to achieve a more abstract description) followed by some form of forward engineering or restructuring. [CHIK, I-CASE] (2) Redesign and development of a system and its data structures from an existing baseline to improve its functionality or configuration. These can be based on reverse engineering products or upon existing documentation and design documents or by other systems analysis methods. [DODM, I-CASE] (3) The process of examining an existing software system (program) and/or modifying it with the aid of

automated tools to improve its future maintainability, upgrade its technology, extend its life expectancy, capture its components in a repository where CASE tools can be used to support it, and increase maintenance productivity. [MCC, I-CASE]

REFERENCE MODEL. A conceptual framework that helps to describe and compare systems. As used here, it describes the set of services an environment provides, but not the architecture of how such services are to be provided. [I-CASE]

REGRESSION TESTING. A software function that performs the rerunning of tests to detect errors spawned by changes or corrections made during software development and maintenance. [I-CASE] Selective retesting to detect faults introduced during modifications of a system or system component, to verify that modifications have not caused unintended adverse effects, or to verify that a modified system or system component still meets its specified requirements.

RELATIONAL DATABASE. A database that allows the linking of different pieces of information. [SWTP]

RELATIONAL DATABASE MANAGEMENT SYSTEM (RDBMS). A database management system in which information is organized in tables. [I-CASE]

RELATIONSHIP. An association between two entity types. [I-CASE]

RELIABILITY. The ability of an item to perform a required function under stated conditions for a stated period of time. [DCT].

REPOSITORY. (1) The place where the software artifacts are kept in their on-line form. Its functions are modeled after those of a librarian; it is the gatekeeper of project products. Artifacts are typically created and modified outside the realm of the repository. [SWTP] (2) The mechanism for defining, storing, accessing, and managing all of the information about an enterprise, its data, and its software systems. [MCC]

REQUIREMENTS ANALYSIS. (1) The process of studying user needs to arrive at a definition of system, hardware, or software requirements (referred to as the Requirements Analysis Phase). (2) The process of studying and refining system, hardware, or software requirements. [I-CASE]

REQUIREMENTS ENGINEERING. Involves all life-cycle activities for identifying user requirements, analyzing the requirements to derive additional requirements, documenting the requirements as a specification, and validating the documented

requirements against user needs. Also refers to processes that support these activities. [STS]

REQUIREMENTS TRACEABILITY. Checking to ensure that user requirements are satisfied by the software system being produced and that all software being produced can be traced back to a requirement. [I-CASE]

REUSABLE SOFTWARE. Software that has application, in whole or in part, to more than one specific function. [DOD-STD-2167A]

REUSE LIBRARY. (1) A library that contains filtered items from many projects, usually within the same domain. A reuse library has a different schema and tooling than a data repository. It usually contains larger grain data. (2) Storage location for software developed in response to the requirements for one application than can be used, in whole or in part, to satisfy the requirements of another application. [DOD-STD-2167A] (3) A central library of reusable software parts that is available to software developers. [DORF]

REVERSE ENGINEERING. The process of analyzing a subject system to identify the system's components and their interrelationships and create representations of the system in another form or at a higher level of abstraction. [I-CASE]

RISK. The probability of occurrence of an undesired event and its impact. [SWTP]

RISK ANALYSIS. The process of identifying risks, determining their magnitude, and identifying areas needing safeguards. Risk analysis is a part of risk management and is synonymous with risk assessment. [I-CASE]

RISK ASSESSMENT. Information provided to identify items at risk, their sources of risk, and their measure of risk. Risk assessment is accomplished through a disciplined and structured process to systematically identify and analyze risks. [SWTP]

RISK MANAGEMENT. Making informed decisions by consciously assessing what can go wrong and the resulting impact. Risk management is accomplished through a continuous set of activities to identify, confront, and resolve risks. [SWTP]

ROBUSTNESS. The extent to which software can continue to operate correctly despite the introduction of invalid inputs. [IEEE]

RUN-TIME ENVIRONMENT. The environment in which applications produced using the I-CASE environment will operate. [I-CASE]

SAFE SOFTWARE. Software that has a level of assurance that the system will not enter a hazardous state. If a hazardous state should occur, the system will get out of it. A hazardous state is one in which an accident or mishap can occur (e.g., when two aircraft come closer than the prescribed safe distance). [SWTP]

SCAFFOLDING. Temporary code that has been developed to interface with one or more modules. This temporary code allows modules to be independently tested while waiting for the permanent interfacing modules to be developed and readied for use.

SCHEDULER. The part of a real-time system that performs task management. [SWTP]

SCHEDULING ALGORITHM. A set of rules that assigns the time at which each task will receive service. [SWTP]

SECURE SOFTWARE. Software that has a level of assurance that the system can enforce a specific security policy. [SWTP]

SENSOR. (1) Equipment that detects and may indicate and/or record objects and activities by means of energy or particles emitted, reflected, or modified by objects. [IEEE] (2) Equipment that detects and indicates terrain configuration, the presence of military targets, and other natural and fabricated objects and activities by means of energy reflected or emitted by such targets or objects. The energy may be nuclear, electromagnetic (including the visible and invisible portions of the spectrum), chemical, biological, thermal, or mechanical (including sound, blast, and earth vibration). [SWTP]

SIMULATION. The representation of selected characteristics of the behavior of one physical or abstract system by another system. In a digital computer system, simulation is done by software; for example, (a) the representation of physical phenomena by means of operations performed by a computer system, (b) the representation of operations of a computer system by those of another computer system. [IEEE]

SOFT REAL-TIME SYSTEM. A system in which a statistical distribution of task service times is acceptable. [SWTP]

SOFTWARE. Computer program instructions and data.

SOFTWARE ARCHITECTURE. The structure and relationships among the components of software. [IEEE]

SOFTWARE BRIDGE. Software that translates from one protocol to another. [SWTP]

SOFTWARE BUG. A software fault. [SWTP]

SOFTWARE DEVELOPMENT CYCLE. (1) The period of time that begins with the decision to develop a software product and ends when the product is delivered. This cycle typically includes a Requirements Phase, Design Phase, Implementation Phase, Test Phase, and sometimes, Installation and Checkpoint Phase. contrast with software life cycle. (2) The period of time that begins with the decision to develop a software product and ends when the product is no longer being enhanced by the developer. (3) Sometimes used as a synonym for software life cycle. *See* product development cycle. [IEEE]

SOFTWARE DEVELOPMENT FILE. An electronic or paper repository for a collection of material pertinent to the development or support of software. Contents typically include (either directly or by reference) design considerations and constraints, design documentation and data, schedule and status information, test requirements, test cases, test procedures, and test results. [DOD-STD-2167A]

SOFTWARE ENGINEERING. (1) The application of science and mathematics by which the capabilities of computer equipment are made useful through computer programs, procedures, and associated documentation. (2) The application of a systematic, disciplined approach to the development, operation, and maintenance of software; (i.e., the application of engineering to software). [IEEE] (3) A science of design, development, implementation, test, evaluation, and maintenance of computer software over its life cycle. [DODD]

SOFTWARE ENGINEERING ENVIRONMENT (SEE). A collection of hardware and software tools enveloped in a process to engineer software. [I-CASE]

SOFTWARE LIFE CYCLE. (1) The period of time that starts when a software product is conceived and ends when the product is no longer available for use. The software life cycle typically includes a Requirements Phase, Design Phase, Implementation Phase, and sometimes, Maintenance and Retirement Phases. Contrast with software development cycle. [IEEE] (2) The series of steps or phases through which the software progresses, usually from conception to retirement. [I-CASE]

SOFTWARE MAINTAINABILITY. The probability that the software can be retained in or restored to a specified status in a prescribed period compatible with mission requirements.

SOFTWARE PROCESS. The sequence of tasks and management techniques used during the creation and evolution of software systems. [SWTP]

SOFTWARE PRODUCT METRICS. The collection of data and analysis from software products for purposes of estimating (a) the quality of the products at each phase of the software development cycle, (b) the reliability of the software, and (c) the size of the software. [SWTP]

SOFTWARE QUALITY. The ability of a software product to satisfy its specific requirements. [DOD-STD-2168]

SOFTWARE REENGINEERING. (1) The examination and alteration of a software system to reconstitute it in a new form and the subsequent implementation of the new form. [IEEE] (2) Reengineering requires understanding the design of existing software that was written in an undisciplined, ad hoc style and then describing the design in a structured form consistent with modern software engineering methods. [SWTP]

SOFTWARE REUSE. (1) The deliberate exploitation of existing software components and related assets to facilitate the development of new software systems. [SWTP] (2) The ability to use existing software components over and over. Reuse is facilitated by the use of an I-CASE repository in which reusable software components can be stored. Software components can consist of requirements, design, code, data, and data models. [I-CASE]

SOFTWARE REUSE TECHNOLOGY. (1) Processes, architectures, and component composition capabilities that facilitate software reuse. (2) The organization, storage, and management of reusable components into repositories. (3) The casual browsing and systematic searching of these repositories to locate potentially useful components. (4) The modification and integration of the components into the evolving system development. [SWTP]

SOFTWARE TOOL. A computer program used to help develop, test, analyze, or maintain another computer program or its documentation (e.g., automated design tool, compiler, test tool, and maintenance tool). [IEEE]

SOFTWARE UPGRADE. Software changes that are made to include significant enhancements and new features that improve the performance and capability, or other attributes, of a software product. Normally, software upgrades are provided to a commercial customer at a cost in addition to the original purchase price. [I-CASE]

SOURCE LINES OF CODE (SLOC). The count of program instructions created by project personnel, excluding comment lines. [I-CASE]

SPECIFICATION. (1) A document that prescribes, in a complete, precise, verifiable manner, the requirements, design, behavior, or other characteristics of a system or system component. (2) The process of developing a specification. (3) A concise statement of a set of requirements to be satisfied by a product, a material, or process indicating, whenever appropriate, the procedure by means of which it may be determined whether the requirements given are satisfied. [IEEE]

SQL (STRUCTURED QUERY LANGUAGE). A standard relational database language. [I-CASE]

STANDARD. An approved, documented, and available set of criteria used to determine the adequacy of an action or object.

STANDARD DATA ELEMENT. A data element that is an Air Force standard (i.e., included in the Air Force Corporate Data Dictionary). [I-CASE]

STRUCTURE CHART. A diagram that identifies modules, activities, or other entities in a system or computer program and shows how larger or more general entities break down into smaller more specific entities. (Synonyms: hierarchy chart, program structure chart). [I-CASE]

SYSTEM SPECIFICATION. A system-level specification. The system specification may be a System/Segment Specification (SSS) (DOD-STD-2167A) or a Design Unit Specification (DUS). [AFC2M2 Guidelines Handbook]

SYSTEM PERFORMANCE ANALYSIS. Measures the performance of computer systems and investigates methods by which that performance can be improved. [I-CASE]

SYSTEM TEST. A test of a product in a total systems environment with other software and hardware product combinations.

TAILOR. To modify or change a set of standards or procedures to better match process or product requirements. [SWTP]

TAILORING. (1) Usually spoken of or referring to acquisition strategy, allows the CDRL item to be written to suit an individual program's needs. No strict format needs to be followed. Basics must be addressed, but the Program Manager has authority to design or plan for specific requirements to meet the optimum balance between need and cost. Tailoring allows flexibility. (2) The process by which individual requirements of standards, DID, and related documents are evaluated to determine their suitability for a specific software production effort, and the modification of those requirements to ensure that each achieves an optimal balance between operational needs and costs. [DOD-HDBK-287A]

TASK. (1) A piece of work assigned to a person or group of persons [Webster]. (2) In an executing software system, a software entity that performs a particular function. Also, the unit of work in the software process that provides a visible management checkpoint. Tasks have entry criteria (preconditions) and exit criteria (post conditions). [SWTP]

TAXONOMY. The general principles of scientific classification. [SWTP]

TECHNOLOGY REFRESHMENT. An addition or substitution (i.e., a new component), upgrade, or update to the I-CASE environment product baseline (hardware and software). [I-CASE]

TEMPLATE. A predefined outline that serves as a pattern for document generation or data input. [I-CASE]

TESTING. The process of exercising or evaluating a system or system component by manual or automated means to verify that it satisfies specified requirements or to identify differences between expected and actual results. [IEEE]

TOTAL QUALITY MANAGEMENT (TQM). (1) The DOD version of a management strategy designed to ensure quality with regard to the performance of management, personnel, and product. The concept originates in the work of Deming and Juran. [SWTP] (2) A management philosophy committed to a focus on continuous improvement of product and services with the involvement of the entire workforce.

TRACEABILITY. The degree to which a relationship can be established between two or more products of the development process, especially products having a predecessor-successor or superior-subordinate relationship to one another. [IEEE]

TRAINING. The level of learning required to adequately perform the responsibilities designated to the function and accomplish the mission assigned to the system.

TRANSVERSE ENGINEERING. The combination of reverse engineering, insertion of design changes, and forward engineering performed in that order. [I-CASE]

TROJAN HORSE. (1) A program that appears to do something useful, yet additionally does something destructive behind one's back. [I-CASE] (2) A computer program with an apparently or actually useful function that contains additional (hidden) functions that surreptitiously exploit the legitimate authorizations of the invoking process to the detriment of security or integrity. [I-CASE] *See also* Virus and Worm.

ULTRA-CRITICAL SOFTWARE. Software that is required to provide service with an extremely low probability of failure (e.g., 10^{-9} failures/hour). [SWTP]

UNIT TEST. The isolated testing of each flow path of code within each module.

USER FRIENDLY. A basic characteristic of a product that simplifies its operation for users and assists users in understanding other product functions. This "ease of use" condition is typically attributed to the quickness with which users can both learn and become productive with a product.

UTILITY. The state or quality of being useful militarily or operationally. Designed for or possessing several useful or practical purposes rather than a single, specialized one.

VALIDATION. (1) The process of evaluating software at the end of the software development process to ensure compliance with software requirements. [IEEE] (2) The process of evaluating software to determine compliance with specified requirements. [DOD-STD-2167A]

VERIFICATION. (1) The process of determining whether the products of a given phase of the software development cycle fulfill the requirements established during the previous phase. (2) Formal proof of program correctness. [IEEE] (3) The act of reviewing, inspecting, testing, checking, auditing, or otherwise establishing and documenting whether items, processes, services, or documents conform to specified requirements. [DCT] (4) The process of evaluating the products of a software development activity to determine the correctness and consistency with respect to the products and standards provided as input to that activity. [DOD-STD-2167A]

VIRTUAL REALITY. Computationally obtained representations of reality (e.g., use of special input and output devices such as head-mounted displays) that create for any operator the illusion of a real world in which the operator can move and function in a natural fashion. [SWTP]

VIRUS. (1) A piece of code that attaches itself to other programs. Once an infected program is run, the virus quickly spreads to system files and other software. (2) A self-propagating Trojan horse, composed of a mission component, a trigger component, and a self-propagating component. [I-CASE] (*See also* Trojan Horse and Worm)

WAIVER. A written authorization to accept an item that, during production or after having been submitted for inspection, is found to depart from the specified requirements, but nevertheless is considered suitable for use "as is" or after rework by an approved method. [I-CASE]

WALKTHROUGH. A technique used to review the design and code of a production effort, which can be conducted throughout the software production process. [I-CASE]

WORK BREAKDOWN STRUCTURE (WBS). A project-oriented family tree, composed of hardware, software, services, and other work tasks, which results from project engineering effort during the development and production of a Defense material item, and which completely defines the project/program. A WBS displays and defines the product (s) to be developed or produced and relates elements of work to be accomplished to each other and to the end product. [I-CASE]

WORM. A program that replicates and spreads, but does not attach itself to other programs. Unlike a virus, it does not require a host to survive and replicate. Worms usually spread within a single computer or over a network of computers. They are not spread through the sharing of programs. (*See also* Trojan Horse and Virus). [I-CASE]

Sources

AFC2M2 — Air Force Command and Control Modernization Methodology, AFC2M2, Guidelines Handbook, 12 March 1991.

ANSI/ASQC — *Quality Assurance Terminology Standards*, ANSI/ASQC A3-1978.

BOEING — System Software Management Group/COMMETRC, Briefing Slides, Boeing Computer Services, 16 December 1991.

CHIK — Chikofsky, E.J, and J.H. Cross II, "Reverse Engineering and Design Recovery: A Taxonomy," *IEEE Software*, January 1990, pp. 13-17.

DCT — *Dictionary of Computer Terms*, eds. D. Downing, M. Covington, Barron's, 1989.

DOC — *Dictionary of Computing*, eds. E.L. Glaser, I.C. Pyle, V. Illingworth, Oxford University Press, 1983.

DODD — Department of Defense, *Management of Computer Resources in Major Defense Systems*, Department of Defense Directive 5000.29, 26 April 1976.

DODM — *Department of Defense Element Standardization Procedures Manual, DOD Manual 8320.1M, (extract)* Draft, 20 December 1991.

DOD-STD-480A — Department of Defense Standard, *Configuration Control Engineering Changes, Deviations and Waivers*, DOD-STD-480A, 1978.

DOD-STD-2167A — *Defense System Software Development*, 29 February 1988.

DOD-STD-2168 — *Defense System Software Quality Program*, 29 April 1988.

DORF — *Standards, Guidelines, and Examples on System and Software Requirements Engineering*, eds. M. Dorfman, R. Thayer, IEEE Computer Society Press Tutorial, 1990.

ELG — Notes and Handouts from ELG meetings.

I-CASE	Source Selection Information—See FAR 3.104 Final RFP Version 24 July 1993 (F01620-91-R-A254).
IEEE	*ANSI/IEEE Software Std 729-1983, IEEE Standard Glossary of Software Engineering Terminology*, 1983.
MARTIN	Martin, J., *Information Engineering, Book I, Introduction*, Prentice Hall, 1989.
MCC	McClure, C., *The Three R's of Software Automation*, Prentice Hall, 1992.
MIL-HDBK-287	*Tailoring Guide for DOD-STD-2167A*, Defense System Software Development, 11 August 1989.
MIL-STD-109B	*Quality Assurance Terms and Conditions*, 4 April 1969.
MIL-STD-973	*Configuration Management*, 17 April 1992.
PUB 1-02	Joint Chiefs of Staff, *Department of Defense Dictionary of Military and Associated Terms*, Washington, D.C., 1989.
STARS	McDonal, C., and S. Redwine, "STARS Glossary: A Supplement to the IEEE Standard Glossary of Software Engineering Terminology Version 3.0," *IDA Paper* (P-1846), January 1986.
STS	DOD Software Technology Strategy, Draft, 31 October 1991.
SWTP	*Software Technology Plan*, Department of Defense, Draft, October 1991.
WEBSTER	*Webster's Third New International Dictionary*, ed. G. P. Babcock, Merriam-Webster Inc., Springfield, MA, 1981.

Bibliography

Ada Joint Program Office. *Common Ada Programming Support Environment (Ada PSE) Interface Set (CAIS) (Revision A).* 6 April 1989.

Archer, T. S. "Managing the Ada Conversion and Integration of Mission Critical Defense Systems." 9th Annual National Conference in Ada Technologies. March 1991.

Assistant Secretary of Defense (Command, Control, Communications, and Intelligence [C3I]). Memorandum: "Plan for Implementation of Corporate Information Management in DOD." 14 January 1991.

Association for Computing Machinery (ACM) Special Interest Group on Ada (SIGAda). *Implementing the DOD-STD-2167 and DOD-STD-2167A Software Organizational Structure in Ada.* August 1990.

Association for Computing Machinery, *Communication*, January 1984.

Baumert, J. and M. McWhinney. *Software Measures and the Capability Maturity Model* (CMU/SEI-92-TR-25, ESC-TR-92-025). Pittsburgh, PA: Carnegie-Mellon University, September 1992.

Berk, K., D. Barrow, and T. Steadman. *Project Management Tools Report.* Hill AFB, UT: Software Technology Support Center, March 1992.

Boehm, B. W., "Ada COCOMO and the Ada Process Model," *Proceedings, Fifth COCOMO Users Group Meeting.* Pittsburgh, PA: Carnegie-Mellon University, October 1989.

Boehm, B.W. *Software Engineering Economics.* Englewood Cliffs, NJ: Prentice-Hall, 1981.

Booch, G. *Software Components with Ada.* Menlo Park, CA.: Benjamin/Cummings Publishing Co., 1987.

Booch, G. *Software Engineering with Ada.* Menlo Park, CA.: Benjamin/Cummings Publishing Co., 1983.

Bowler, O. et al. *Requirements Analysis & Design Tools Report.* Hill AFB, UT: Software Technology Support Center, April 1992.

Carleton, A. et al. *Software Measurement for DOD Systems: Recommendations for Initial Core Measures* (CMU/SEI-92-TR-19, ESC-TR-92-019). Pittsburgh, PA: Carnegie-Mellon University, September 1992.

Carnegie-Mellon University/Software Engineering Institute. *Software Process Assessment.* Pittsburgh, PA: Carnegie-Mellon University, September 1990.

Carnegie-Mellon University/Software Engineering Institute. *Software Metrics* (SEI-CM-12-1.1). Pittsburgh, PA: Carnegie-Mellon University, December 1988.

Cohen, S. G. *Ada 9X Issues—Reuse Study* (Draft). Pittsburgh, PA: Carnegie-Mellon University, 1989.

Conn, R. "Overview of the DOD Ada Software Repository." *Dr. Dobb's Journal,* February 1986.

Crosby, D., G. Petersen, and R. Sorensen. *Documentation Tools Report.* Hill AFB, UT: Software Technology Support Center, March 1992.

Deputy Secretary of Defense. Memorandum: "Strengthening Technology and Acquisition Functions." 12 August 1991.

Donohue, P. *Hartstone Benchmark User's Guide,* Version 1.0, (TR SEI-90-UG-1, ADA 235740). Pittsburgh, PA: Carnegie-Mellon University, 1990.

Dyson, P. B. *Metrics Application Plan for the Take Charge and Move Out (TACAMO) Message Processing System: Technical Report.* (Prepared for the Naval Air Development Center, Warminster, PA, under Contract Number: N62269-86-C-0415), 27 April 1989.

Evaluation and Validation Guidebook, Version 3.0. (NTIS A 236 494). E&V, 1991(a).

Evaluation and Validation Reference Manual, Version 3.0. (NTIS A 236 697). E&V 1991(b).

EVB Software Engineering, Inc. "Creating Reusable Ada Software," *Proceedings of the Conference on Software Reusability and Maintainability.* Tysons Corner, VA: The National Institute for Software Quality and Productivity, Inc., March 1987.

Feiler, P. and G. Downey. *Tool Version Management Technology: A Case Study* (CMU/SEI-90-TR-26, Ada 235639). Pittsburgh, PA: Carnegie-Mellon University, 1990.

FIPS-PUBS 127-1. *Database Language SQL*, 2 February 1990 (incorporates ANSI X.3 135-1989, *Database Language—SQL With Integrity Enhancement* and ANSI X.3 168-1989, *Database Language—Embedded SQL*).

Florac, W. *Software Quality Measurement: A Framework for Counting Problems and Defects* (CMU/SEI-92-TR-22, ESC-TR-92-022). Pittsburgh, PA: Carnegie-Mellon University, September 1992.

Giallombardo, R. J. *Effort and Schedule Estimating Models for Ada Software Developments*. MITRE, May 1992.

Goethert, W., E. Bailey, and M. Busby. *Software Effort and Schedule Measurement: A Framework for Counting Staff-Hours and Reporting Schedule Information* (CMU/SEI-92-TR-21, ESC-TR-92-021). Pittsburgh, PA: Carnegie-Mellon University, September 1992.

Guidelines for Professional Programmers. New York, NY: Van Nostrand, Reinhold, 1989.

Hanrahan, B. et al. *Software Engineering Environment Report*. Hill AFB, UT: Software Technology Support Center, March 1992.

Hefley, W. E., J.T. Foreman, C.B. Engle, Jr., and J.B. Goodenough. *Ada Adoption Handbook: A Program Manager's Guide*, Version 2.0 (CMU/SEI-92-TR-29, ESC-TR-92-029). Pittsburgh, PA: Carnegie-Mellon University, October 1992.

Hitchon et al., *Introduction to CAIS (MIL-STD-1838A)*, 1989.

Humphrey, W. S. *Managing the Software Process*. Reading, MA: Addison-Wesley, 1989.

Humphrey, W. S. *Characterizing the Software Process: A Maturity Framework*, (CMU/SEI-87-TR-11, ESD-TR-87-112). Pittsburgh, PA: Carnegie-Mellon University, June 1987.

Humphrey, W. S. and W.L. Sweet. *A Method for Assessing the Software Engineering Capability of Contractors* (CMU/SEI-87-TR-23, ESD-TR-87-186). Pittsburgh, PA: Carnegie-Mellon University, September 1987.

ITT Research Institute. *Available Ada Bindings*. January 1992.

ITT Research Institute. *Estimating the Cost of Ada Software Development, Test Case Study*. April 1989

Jones, C. *Applied Software Measurement: Assuring Productivity and Quality.* New York, NY: McGraw-Hill, 1991.

Kile, R. L. *Revised Intermediate COCOMO (REVIC) Software Cost Estimating Model User's Manual*, Version 9.0. February 1991.

Law, D. "Parallel Ada in Simulation Systems." *Defense Electronics*, Vol. 24, No. 11. November 1992, pp. 35-37.

Lesslie, P. A., R. O. Chester, and M. F. Theofanos. *Guidelines Document for Ada Reuse and Metrics* (Draft). Oak Ridge, TN: Martin Marietta Energy Systems, Inc., 1988.

McDonnell Douglas Missile System Company. *Common Ada Missile Packages—Leading the Way in Software Reuse* (videotapes). St. Louis, MO: McDonnell Douglas, 1991.

McNicholl, D. G. et al. *Common Missile Packages (CAMP)*, Vol. I: *Overview and Commonality Study Results* (AFATL TR-85-93). St. Louis, MO: McDonnell Douglas, 1986.

MIL-HDBK 287. "A Tailoring Guide for DOD-STD-2167A." 29 February 1988.

MIL-STD-1838A. "Common Ada Programming Support Environment (Ada PSE) Interface Set." 30 September 1989.

Musgrove, R. G. *Workshop on Commonality in Computing for NASA Flight Systems.* Houston, TX: Lyndon B. Johnson Space Center, 1987.

National Aeronautics and Space Administration. *Software Engineering Laboratory (SEL) Guidebook.*

Naval Air Warfare Center. *Next Generation Computer Resources Reference Model for Project Support Environments*, Technical Report NAVCADWAR 92023-70, 1993.

Naval Ocean Systems Center (NOSC). *Guideline for NOSC Ada Programmers: An Update to SPC's Ada Quality and Style*, Version 1.0. 30 September 1991.

Naval Underwater Systems Center (NUSC) Software Metrics. *Ada Style Guide.* 10 May 1991.

Park, R. *Software Size Measurement: A Framework for Counting Source Statements* (CMU/SEI-92-TR-20, ESC-TR-92-20). Pittsburgh, PA: Carnegie-Mellon University, September 1992.

Price, G. et al. *Test Preparation, Execution, and Analysis Tools Report, Rev-A.* Hill AFB, UT: Software Technology Support Center, April 1992.

Price, G. et al. *Source Code Static Analysis Tools Report, Rev-A.* Hill AFB, UT: Software Technology Support Center, April 1992.

Rozum, J. et al. *Software Measurement Concepts for Acquisition Program Managers* (CMU/SEI-92-TR-11, ESD-TR-92-11). Pittsburgh, PA: Carnegie-Mellon University, June 1992.

San Antonio I, Panel I. *Software Metrics Implementation Final Report.* 11 December 1991.

San Antonio I, Panel VII. *JLC Software Workshop Final Report.* 1 February 1991.

SEER Technologies Division. *System Evaluation and Estimation of Resources* (SEER™), *User's Manual.* Marina del Ray, CA: Calorath Associates, Inc., 15 March 1991.

Shlaer, S. and S. Miller. "An Object-Oriented Approach to Domain Analysis." *Software Engineering Notes*, Vol. 14, No. 5. November 1989.

Sittenaur, C., M. Olsen, and D. Murdock. *Reengineering Tools Report, Rev-B.* Hill AFB, UT: Software Technology Support Center, July 1992.

Software Architecture Sizing and Estimating Tool (SASET), SASET User's Guide, Version 3.0. September 1992.

Software Cost Estimation Study: Performance Measurement Enhancement (Software Performance Measurement Model [SPMM]), User's Manual. August 1990.

Software Standards and Procedures Manual for the AN/BSY-2 Submarine Combat System. 28 September 1990.

Software Standards and Procedures Manual for the CCS Mk 2 Submarine Combat Control System. 5 October 1989.

Software Technology Support Center. *Software Management Guide*, third printing. Hill AFB: Ogden Air Logistics Center, 1992.

U.S. Air Force. *Air Force Systems Command Software Quality Indicators: Management Quality Oversight* (AFSCP 800-14). January 1987.

U.S. Air Force. *Air Force Systems Command Software Management Indicators: Management Insight* (AFSCP 800-43). January 1986.

U.S. Department of Commerce, National Institute of Standards and Technology. *Application Portability Profile (APP): The U.S. Government's Open System Environment Profile OSE/1 Version 1.0.* Washington, D.C.: Government Printing Office, 1991.

U.S. Department of Commerce, National Institute of Standards and Technology. *Reference Model for Frameworks of Software Engineering Environments*, (NIST Special Publication 500-201/Technical Report ECMA TR/55, second edition). Washington, D.C.: Government Printing Office, 1991.

U.S. Department of the Navy. *Interim Department of the Navy Policy on Ada.* 24 June 1991.

Van Verth, P. *A Concept Study for a National Software Engineering Database* (CMU/SEI-92-TR-23, ESC-TR-92-003). Pittsburgh, PA: Carnegie-Mellon University, July 1992.

Weiderman, N. *Ada Adoption Handbook, Compiler Evaluation and Selection*, Version 1.0 (CMU/SEI 89-TR-13, ESD-TR-89-12). Pittsburgh, PA: Carnegie-Mellon University, 1989.

Index

END